**CCC 2014**

2014 - J1

def main():

ang1 = int(raw\_input())

ang2 = int(raw\_input())

ang3 = int(raw\_input())

total = ang1 + ang2 + ang3

if total != 180:

return "Error"

if ang1 == ang2 and ang1 == ang3:

return "Equilateral"

elif ang1 == ang2 or ang1 == ang3 or ang2 == ang3:

return "Isosceles"

else:

return "Scalene"

if \_\_name\_\_ == "\_\_main\_\_":

print(main())

2014 - J2

def main():

number = int(raw\_input())

votes = raw\_input()

Blist = votes.split("A")

Bs = "".join(Blist)

BNumber = len(Bs)

ANumber = number - BNumber

if ANumber == BNumber:

return "Tie"

elif ANumber > BNumber:

return "A"

else:

return "B"

if \_\_name\_\_ == "\_\_main\_\_":

print(main())

2014 - J3

def getDices():

number = int(raw\_input())

dices = []

for i in range(number):

picks = raw\_input().split(" ")

ant\_p = int(picks[0])

dav\_p = int(picks[1])

dices.append([ant\_p, dav\_p])

return dices

def getResult(dices):

ant\_score = 100

dav\_score = 100

for d in dices:

if d[0] > d[1]:

dav\_score = dav\_score - d[0]

elif d[0] < d[1]:

ant\_score = ant\_score - d[1]

else:

pass

return (ant\_score, dav\_score)

if \_\_name\_\_ == "\_\_main\_\_":

dices = getDices()

result = getResult(dices)

print(result[0])

print(result[1])

2014 - J4

import os

friendsList = []

removeBaseList = []

def initialize():

global removeBaseList, friendsList

numberOfFriends = int(raw\_input())

friendsList = [int(i+1) for i in range(numberOfFriends)]

numberOfRounds = int(raw\_input())

for i in range(numberOfRounds):

removeBaseList.append(int(raw\_input()))

def removeFriends(base, alist):

global friendsList

if base > len(alist):

return

curr\_idx = base - 1

count = 1

remove\_indices = []

while curr\_idx < len(alist):

remove\_indices.append(curr\_idx)

count += 1

curr\_idx = count \* base - 1

newList = []

for i in range(len(friendsList)):

if i not in remove\_indices:

newList.append(friendsList[i])

friendsList = newList

def j4\_2014():

global friendsList

os.system('clear')

for item in removeBaseList:

removeFriends(item, friendsList)

for ff in friendsList:

print ff

if \_\_name\_\_ == "\_\_main\_\_":

initialize()

j4\_2014()

2014 - J5

numberOfStudents = 0

group1 = []

group2 = []

def initialize():

global numberOfStudents, group1, group2

numberOfStudents = int(raw\_input())

group1 = raw\_input().split()

group2 = raw\_input().split()

def checkPairs():

mygroup1 = group1[:]

mygroup2 = group2[:]

while mygroup1 != []:

name = mygroup1[0]

if not isMatched(name):

return False

else:

name2 = mygroup2[0]

mygroup1.pop(0)

mygroup1.pop(mygroup1.index(name2))

mygroup2.pop(0)

mygroup2.pop(mygroup2.index(name))

return True

def isMatched(name1):

idx1 = group1.index(name1)

name2 = group2[idx1]

if name1 == name2:

return False

idx2 = group1.index(name2)

name3 = group2[idx2]

if name1 == name3:

return True

else:

return False

if \_\_name\_\_ == "\_\_main\_\_":

initialize()

if checkPairs():

print("good")

else:

print("bad")

**CCC 2015**

2015 - J3

allLetters = "abcdefghijklmnopqrstuvwxyz"

allVowels = "aeiou"

def convertCharacter(ch):

if ch in allVowels:

return ch

letters = ch

curr\_pos = allLetters.index(ch)

prev\_vowel\_pos = findVowelBefore(curr\_pos)

next\_vowel\_pos = findVowelAfter(curr\_pos)

if prev\_vowel\_pos == curr\_pos:

letters = letters + allLetters[next\_vowel\_pos]

elif next\_vowel\_pos == curr\_pos:

letters = letters + allLetters[prev\_vowel\_pos]

else:

prev\_diff = curr\_pos - prev\_vowel\_pos

next\_diff = next\_vowel\_pos - curr\_pos

if prev\_diff < next\_diff:

letters = letters + allLetters[prev\_vowel\_pos]

elif prev\_diff > next\_diff:

letters = letters + allLetters[next\_vowel\_pos]

else:

letters = letters + allLetters[prev\_vowel\_pos]

next\_consonant = allLetters[getNextConsonant(curr\_pos)]

letters += next\_consonant

return letters

def findVowelBefore(curr\_pos):

prev\_pos = curr\_pos

for idx in range(curr\_pos, -1, -1):

if allLetters[idx] in allVowels:

prev\_pos = idx

break

return prev\_pos

def findVowelAfter(curr\_pos):

next\_pos = curr\_pos

for idx in range(curr\_pos, len(allLetters)):

if allLetters[idx] in allVowels:

next\_pos = idx

break

return next\_pos

def getNextConsonant(curr\_pos):

next\_pos = curr\_pos

for idx in range(curr\_pos+1, len(allLetters)):

if allLetters[idx] not in allVowels:

next\_pos = idx

break

return next\_pos

def translateWord(word):

newWord = ""

for ch in word:

if ch not in allVowels:

newCh = convertCharacter(ch)

newWord += newCh

else:

newWord += ch

return newWord

if \_\_name\_\_ == "\_\_main\_\_":

word = raw\_input()

print(translateWord(word))

2015 - J4

messageRecord = {}

result = {}

def initialize():

global messageRecord

numberOfMessages = int(raw\_input())

curr\_time = 0

for idx in range(numberOfMessages):

line = raw\_input()

messageType = line.split()[0]

number = int(line.split()[1])

if messageType == "W":

curr\_time += number - 1

elif messageType == "R":

curr\_time += 1

if number in messageRecord:

messageRecord[number]["R"].append(curr\_time)

else:

messageRecord[number] = {}

messageRecord[number]["R"] = []

messageRecord[number]["S"] = []

messageRecord[number]["R"].append(curr\_time)

else:

curr\_time += 1

if number in messageRecord:

messageRecord[number]["S"].append(curr\_time)

else:

messageRecord[number] = {}

messageRecord[number]["R"] = []

messageRecord[number]["S"] = []

messageRecord[number]["S"].append(curr\_time)

def matchRecords(rlist, slist):

if len(rlist) > len(slist):

return -1

total = 0

for idx in range(len(rlist)):

rtime = rlist[idx]

stime = slist[idx]

diff = slist[idx] - rlist[idx]

total += diff

return total

def getSummary():

global result

for friend in messageRecord:

R\_res = messageRecord[friend]["R"]

S\_res = messageRecord[friend]["S"]

friend\_result = matchRecords(R\_res, S\_res)

result[friend] = friend\_result

def summaryPrintOut():

#print messageRecord

for friend in result:

print friend, result[friend]

if \_\_name\_\_ == "\_\_main\_\_":

initialize()

getSummary()

summaryPrintOut()

2015 - J5

numberOfPies = 1

numberOfPeople = 1

collection = []

def initialize():

global numberOfPies, numberOfPeople

numberOfPies = int(raw\_input())

numberOfPeople = int(raw\_input())

def distributePies():

global collection

currList = [1 for i in range(numberOfPeople)]

restPies = numberOfPies - numberOfPeople

distributeHelper(currList, restPies)

def distributeHelper(currList, restPies):

global collection

if restPies == 0:

if currList not in collection:

collection.append(currList)

print(len(collection))

else:

newRestPies = restPies - 1

newList = sorted(currList)

combinations = findAllCombinations(newList)

for alist in combinations:

distributeHelper(alist, newRestPies)

def findAllCombinations(testList):

combinations = []

distinctList = set(testList)

for num in distinctList:

idx = testList.index(num)

newList = testList[:]

newList[idx] = newList[idx] + 1

newList.sort()

combinations.append(newList)

return combinations

def findAllNewLists(alist):

newList = alist[:]

newList.sort()

collection = []

mySet = set(alist)

for item in mySet:

idx = newList.index(item)

aNewList = newList[:]

aNewList[idx] = newList[idx] + 1

aNewList.sort()

collection.append(aNewList)

return collection

if \_\_name\_\_ == "\_\_main\_\_":

initialize()

distributePies()

print(len(collection))

**CCC 2016**

2016 - J3

def j3\_2016(word):

if isPalindrome(word):

return len(word)

maxLength = 1

for i in range(len(word)-1):

length = longestPalindrome(word[i:])

if length > maxLength:

maxLength = length

return maxLength

def longestPalindrome(word):

if len(word) == 1:

return 1

for i in range(len(word), 1, -1):

if isPalindrome(word[:i]):

return i

return 1

def isPalindrome(word):

if len(word) == 1:

return True

for i in range(len(word)/2):

front = word[i]

back = word[len(word)-1-i]

if front != back:

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

testWord = raw\_input()

print(j3\_2016(testWord))

2016 - J5

question = 1

numberOfCitizens = 0

dmoSpeeds = []

pegSpeeds = []

"""

1: the minimum total speed out of all the pairs

2: the maximum total speed out of all the pairs

"""

def initialize():

global question, numberOfCitizens, dmoSpeeds, pegSpeeds

question = int(raw\_input())

numberOfCitizens = int(raw\_input())

line = raw\_input().split()

dmoSpeeds = [int(i) for i in line]

line = raw\_input().split()

pegSpeeds = [int(i) for i in line]

dmoSpeeds.sort()

pegSpeeds.sort()

def getMinimumSpeed():

total = 0

for idx in range(numberOfCitizens-1, -1, -1):

s1 = dmoSpeeds[idx]

s2 = pegSpeeds[idx]

total += max(s1, s2)

return total

def getMaximumSpeed():

total = 0

for idx in range(numberOfCitizens-1, -1, -1):

s1 = dmoSpeeds[idx]

s2 = pegSpeeds[numberOfCitizens-idx-1]

total += max(s1, s2)

return total

if \_\_name\_\_ == "\_\_main\_\_":

initialize()

if question == 1:

print(getMinimumSpeed())

else:

print(getMaximumSpeed())

2016 - J4

def getArrivalTime(hours, minutes):

if hours < 5:

return (hours+2, minutes)

elif hours < 7:

minutes\_zone\_left = 60 \* (7 - hours) - minutes

minutes\_left = 120 - minutes\_zone\_left

minutes\_in\_zone = minutes\_left \* 2

if minutes\_in\_zone < 180:

new\_hours = 7 + minutes\_in\_zone / 60

new\_minutes = minutes\_in\_zone % 60

return (new\_hours, new\_minutes)

else:

minutes\_out\_zone = (minutes\_in\_zone - 180) / 2

new\_hours = 10 + minutes\_out\_zone / 60

new\_minutes = minutes\_out\_zone % 60

return (new\_hours, new\_minutes)

elif hours < 10:

minutes\_in\_zone = (10 - hours) \* 60 - minutes

minutes\_out\_zone = (240 - minutes\_in\_zone) / 2

new\_hours = 10 + minutes\_out\_zone / 60

new\_minutes = minutes\_out\_zone % 60

return (new\_hours, new\_minutes)

elif hours < 13:

return (hours+2, minutes)

elif hours < 15:

minutes\_zone\_left = 60 \* (15 - hours) - minutes

minutes\_left = 120 - minutes\_zone\_left

minutes\_in\_zone = minutes\_left \* 2

new\_hours = 15 + minutes\_in\_zone / 60

new\_minutes = minutes\_in\_zone % 60

return (new\_hours, new\_minutes)

elif hours < 19:

minutes\_in\_zone = (19 - hours) \* 60 - minutes

minutes\_out\_zone = (240 - minutes\_in\_zone) / 2

new\_hours = 19 + minutes\_out\_zone / 60

new\_minutes = minutes\_out\_zone % 60

return (new\_hours, new\_minutes)

elif hours < 22:

return (hours+2, minutes)

else:

new\_hours = hours - 22

new\_minutes = minutes

return (new\_hours, new\_minutes)

if \_\_name\_\_ == "\_\_main\_\_":

time = raw\_input()

hours = int(time.split(":")[0])

minutes = int(time.split(":")[1])

new\_hours, new\_minutes = getArrivalTime(hours, minutes)

if new\_hours < 10:

new\_hours = "0" + str(new\_hours)

else:

new\_hours = str(new\_hours)

if new\_minutes < 10:

new\_minutes = "0" + str(new\_minutes)

else:

new\_minutes = str(new\_minutes)

print new\_hours + ":" + new\_minutes

**CCC 2017**

2017 - J3

start\_coord = raw\_input().split()

xs = int(start\_coord[0])

ys = int(start\_coord[1])

end\_coord = raw\_input().split()

xe = int(end\_coord[0])

ye = int(end\_coord[1])

batt = int(raw\_input())

def j3\_2017():

global xs, ys, xe, ye, batt

if batt == 0 and xs == xe and ys == ye:

return True

elif batt == 0:

return False

else:

return move\_car(xs-1, ys, xe, ye, batt) or move\_car(xs+1, ys, xe, ye, batt) \

or move\_car(xs, ys-1, xe, ye, batt) or move\_car(xs, ys+1, xe, ye, batt)

def move\_car(x1, y1, x2, y2, bat):

bat = bat - 1

if bat == 0:

if x1 == x2 and y1 == y2:

return True

else:

return False

else:

x\_new = x1 + 1

if move\_car(x\_new, y1, x2, y2, bat):

return True

x\_new = x1 - 1

if move\_car(x\_new, y1, x2, y2, bat):

return True

y\_new = y1 + 1

if move\_car(x1, y\_new, x2, y2, bat):

return True

y\_new = y1 - 1

if move\_car(x1, y\_new, x2, y2, bat):

return True

return False

if \_\_name\_\_ == "\_\_main\_\_":

if j3\_2017():

print("Y")

else:

print("N")

Small Practices

"""

compute all the mnemonics for a phone number

the user input a valid phone number, output a list of combinations

1 - 1

2 - ABC

3 - DEF

4 - GHI

5 - JKL

6 - MNO

7 - PQRS

8 - TUV

9 - WXYZ

0 - 0

"""

numberMap = {

"0": "0",

"1": "1",

'2': "ABC",

'3': 'DEF',

'4': 'GHI',

'5': 'JKL',

'6': 'MNO',

'7': 'PQRS',

'8': 'TUV',

'9': 'WXYZ'

}

def findAllMnemonics(numberString):

retList = []

numbers = list(numberString)

partial = numbers[:]

phoneMnemonicHelper(numbers, 0, partial, retList)

return retList

def phoneMnemonicHelper(numbers, position, partial, retList):

if position == len(numbers):

retList.append("".join(partial))

else:

mappedChar = numberMap[numbers[position]]

for ch in mappedChar:

newPartial = partial[:]

newPartial[position] = ch

phoneMnemonicHelper(numbers, position+1, newPartial, retList)

if \_\_name\_\_ == "\_\_main\_\_":

numbers = raw\_input()

print(findAllMnemonics(numbers))