random.seed(0)

def sameDate(numPeople, numSame):
    possibleDates = range(366)
    birthdays = [0]*366
    for p in range(numPeople):
        birthDate = random.choice(possibleDates)
        birthdays[birthDate] += 1
    return max(birthdays) >= numSame

def birthdayProb(numPeople, numSame, numTrials):
    numHits = 0
    for t in range(numTrials):
        if sameDate(numPeople, numSame):
            numHits += 1
    print 'Est. probability of', numSame, 'shared birthdays =',
    print numHits/float(numTrials)

birthdayProb(150, 4, 10000)

def maxPeople(numPeople):
    possibleDates = range(366)
    birthdays = [0]*366
    for p in range(numPeople):
        birthDate = random.choice(possibleDates)
        birthdays[birthDate] += 1
    return max(birthdays)

def maxBirthdayProb(numPeople, numTrials):
    maxVals = []
    for t in range(numTrials):
        maxVals.append(maxPeople(numPeople))
    print max(maxVals)
    print 'Print mean maximum number of shared birthdays =',
    print sum(maxVals)/float(numTrials)
class intDict(object):
    """A dictionary with integer keys""

    def __init__(self, numBuckets):
        """Create an empty dictionary""
        self.numBuckets = numBuckets
        self.buckets = []
        for i in range(numBuckets):
            self.buckets.append([])

    def addEntry(self, dictKey, dictVal):
        """Assumes dictKey an int. Adds an entry.""
        hashBucket = self.buckets[dictKey%self.numBuckets]
        for i in range(len(hashBucket)):
            if hashBucket[i][0] == dictKey:
                hashBucket[i] = (dictKey, dictVal)
                return
        hashBucket.append((dictKey, dictVal))

    def getValue(self, dictKey):
        """Assumes dictKey an int. Returns entry associated with the key dictKey""
        hashBucket = self.buckets[dictKey%self.numBuckets]
        for e in hashBucket:
            if e[0] == dictKey:
                return e[1]
        return None

    def __str__(self):
        res = '{
        for b in self.buckets:
            for t in b:
                res = res + str(t[0]) + ':' + str(t[1]) + ','
        return res[:-1] + '}'  #res[:-1] removes the last comma

    def printBuckets(self):
        print '\n', 'The buckets are:
        for hashBucket in D.buckets:
            print ' ', hashBucket

def hashStr(s, tableSize):
    number = ''
    for c in s:
        number = number + str(ord(c))
    print 'Numerical representation =', number
    print 'Hash value =', int(number)%tableSize

    keys = ('MIT', 'Harvard', 'New England Patriots', 'Cambridge', '2')
    for k in keys:
        hashStr(k, 20)