def rollDie():
    """returns a random int between 1 and 6"""
    return random.choice([1, 2, 3, 4, 5, 6])

def runSim(goal, numRolls):
    total = 0
    for i in range(numRolls):
        result = ''
        for j in range(len(goal)):
            result += str(rollDie())
        if result == goal:
            total += 1
    probability = round(total/float(numRolls), 5)
    print 'Actual probability =', round(1.0/(6**len(goal)),5)
    print 'Estimated Probability =', probability

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 'Probability of a shared birthday =', numHits/float(numTrials)

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 'Print mean maximum number of shared birthdays =',\
              sum(maxVals)/float(numTrials)
    maxVals.sort()
    print 'Print median number of shared birthdays =',\
          maxVals[len(maxVals)%2]
    print max(maxVals)
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."""
        bucket = self.buckets[dictKey % self.numBuckets]
        for i in range(len(bucket)):
            if bucket[i][0] == dictKey:
                bucket[i] = (dictKey, dictVal)
                return
        bucket.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] + '}'
    def printBuckets(self):
        print '\n', 'The buckets are:
        for hashBucket in D.buckets:
            print '  ', hashBucket
D = intDict(29)
for i in range(20):
    # choose a random int in range(10**5)
    val = range(10**5)
    key = random.choice(val)
    D.addEntry(key, i)
print 'The value of the intDict is:'
print D
D.printBuckets()

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)