A = Die('A', 1.0/5.0)
B = Die('B', 1.0/6.0)
C = Die('C', 1.0/7.0)
dice = (A, A, A, B, B, C)

# Compute priors
numA, numB, numC = 0, 0, 0
for c in dice:
    if c == A:
        numA += 1
    elif c == B:
        numB += 1
    else:
        numC += 1
priorA = float(numA)/len(dice)
priorB = float(numB)/len(dice)
priorC = float(numC)/len(dice)
badPriorA = priorC
badPriorB = priorB
badPriorC = priorA

for numRolls in (2, 20, 200, 2000, 20000):
    random.seed(1)
    print'\n****Number of rolls is', numRolls, '****'
    print 'Expected fraction for A =', round(A.getProb6(), 3),
    'for B =', round(B.getProb6(), 3),
    'for C =', round(C.getProb6(), 3)
    print 'With priors for A =', round(priorA, 3),
    'for B =', round(priorB, 3),
    'for C =', round(priorC, 3)
    die = random.choice(dice)
    probA, probB, probC, fracSix = calcProbs(A, B, C, priorA,
                                           priorB, priorC,
                                           numRolls, False)

    print 'Fraction of sixes =', fracSix
    print 'Probability of A is', probA
    print 'Probability of B is', probB
    print 'Probability of C is', probC

    random.seed(1)
    print 'With priors for A =', round(badPriorA, 3),
    'for B =', round(badPriorB, 3),
    'for C =', round(badPriorC, 3)
    die = random.choice(dice)
    probA, probB, probC, fracSix = calcProbs(A, B, C, badPriorA,
                                           badPriorB, badPriorC,
                                           numRolls, False)

    print 'Fraction of sixes =', fracSix
    print 'Probability of A is', probA
    print 'Probability of B is', probB
    print 'Probability of C is', probC
    print 'The die is actually of type', die
    raw_input('Hit enter to continue.')
def juneProb(numTrials):
    juneBirths = []
    for trial in range(numTrials):
        june = 0.0
        for i in range(446):
            if random.randint(1,12) == 6:
                june += 1.0
        juneBirths.append(june)
    mean = round(sum(juneBirths)/len(juneBirths), 2)
    sd = round(stdDev(juneBirths), 2)
    print 'Expected number of births in June =', mean, ' +/ - ' + str(2*sd)

def anyProb(numTrials):
    maxMonth = []
    for trial in range(numTrials):
        months = [0.0]*12
        for i in range(446):
            months[random.randint(0,11)] += 1
        maxMonth.append(max(months))
    mean = round(sum(maxMonth)/len(maxMonth), 2)
    sd = round(stdDev(maxMonth), 2)
    print 'Expected number of births in month with most births =', mean, ' +/ - ' + str(2*sd)