Alexander Fielding

EE381 M/W

February 2nd 2018

Problem 1

**INTRODUCTION**

The purpose of this simulation was to continuously roll a pair of dice until a sum of 7 was achieved. The number of rolls is recorded and the simulation continues until 10,000 instances have been recorded. We then compare the results to determine what the most occurring number of required rolls to eventually roll a sum of 7.

**METHODOLOGY**

2 functions are implemented, one to handle the rolling of the dice and another to handle repeating the dice roll until the result is a 7. When the result of 7 occurs, the number of rolls required to reach that roll is returned. This result is stored in a list to be used for the pmf plot.

**RESULTS AND CONCLUSIONS: Graphs (PMF), Probability values, etc**

**APPENDIX:**

import random

#Alexander Fielding

#EE 381 Project 1-1

#Globals

N = 10000

#Rolls 2 dice and returns the result of the dice roll

def diceroll():

num1 = random.randint(0,6)

num2 = random.randint(0,6)

return num1 + num2

#calls diceroll until the result is 7 and returns the number of attempts that occurred

def find7():

found = False

count = 0

while (found != True):

count = count + 1

myseven = diceroll()

if (myseven == 7):

found = True

return count

#Main

attempts = []

while(len(attempts) < 10000):

num = find7()

attempts.append(num)

**Problem 2**

**INTRODUCTION**

In this simulation, we were tasked in determining the probability of exactly 50 coins resulting in heads after tossing 100 coins.

**METHODOLOGY**

This simulation was implemented by utilizing the python random library and using this library to implement a coinToss() method that would serve as a simulation for tossing a single coin. The method would be called 100 times to act as 100 coins being tossed. If the occurrence of heads is exactly 50, we store the simulation result and then repeat the process 10,000. Each time the desired result occurs we increment the total number of “perfect tosses” (exactly 50 heads).

**RESULTS AND CONCLUSIONS:**

|  |  |
| --- | --- |
| Probability of 50 heads in tossing 100 fair coins |  |
| ANS | P = 7.66 |

**APPENDIX:**

import random

#Alexander Fielding

#EE 381 Project 1

#Globals

N = 10000

def coinToss():

num1 = random.randint(0,1)

if(num1 == 1):

return "heads"

else:

return "tails"

**#MAIN**

while(i < 10000):

j = 0

while(j < 100):

mystr = coinToss()

j = j + 1

if( mystr == "heads"):

headcount = headcount + 1

else:

tailcount = tailcount + 1

if(headcount == 50):

totalheads = totalheads + 1

i = i + 1

headcount = 0

print("Number of times exactly 50 coin tosses out of 100 resulted in heads: ", totalheads)

**Problem 3**

**INTRODUCTION**

The simulation is used to determine the probability of drawing a “four of a kind”, 4 cards of the same value, from a 52-card deck.

**METHODOLOGY**

For simplicity, the suits and other card details are ignored and the simulation is based purely on the card’s values. Therefore if 4 of the same number are randomly selected from the “deck” the result is a 4 of a kind and is recorded.

**RESULTS AND CONCLUSIONS:**

|  |  |
| --- | --- |
| **Probability of 4 of a kind** |  |
| **ANS** | **0.019 %** |

**APPENDIX: CODE GOES HERE**

import random

def draw():

alist = [1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,8,9,9,9,9,10,10,10,10,11,11,11,11,12,12,12,12,13,13,13,13]

random.shuffle(alist)

hand = random.sample(alist,5)

return hand

count = 0

results = 0

kind = 0

draws = 1

success = 0;

while(count <= 100000):

count += 1

hand = draw()

first = hand[0]

kind = 0

for i in hand:

if i == first:

kind += 1

if kind == 4:

print(hand)

print("4 of a kind!")

print("number of draws: ",draws)

success += 1;

else:

draws += 1

print("number of 4 of a kinds drawn",success)

**Problem 4**

**INTRODUCTION**

There are 26^4 possible combinations for 4 letter lowercase passwords using the English language. Our task was determining the odds of randomly selected a 4-letter passcode that happens to coincide in a spate list of passcodes using varying sample sizes.

**METHODOLOGY**

To implement this simulation, I utilized a List storing the 26 lower case characters and then used the random class to continuously generate random passcodes into a new list storing the passcodes. A personal passcode is generated and then compared to all the elements in the passcode list to determine if there is a match

**RESULTS AND CONCLUSIONS:**

|  |  |
| --- | --- |
| m = 10^5  Prob. that at least one of the words matches the password | **p = 34/10000** |
| m = 10^6  Prob. that at least one of the words matches the password | **p = 257/100000** |
| p = 0.5  Approximate number of words in the list | **m = 1061** |

**APPENDIX: CODE GOES HERE**

import random

#Global

alpha = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n"

,"o","p","q","r","s","t","u","v","w","x","y","z"]

#func

def linsearch(alist,pswd):

for i in range(len(alist)):

if(alist[i] == pswd):

return True

return False

#main

pswdlist = []

#create a list of 100,000 passwords

random.shuffle(alpha)

while(len(pswdlist) < 10000):

newpswd = ""

chars = (random.sample(alpha,4))

for c in chars:

newpswd += c

pswdlist.append(newpswd)

print("list made")

#make myself a random password

amount = 0

count = 0

while(count < 1000):

count += 1

chars = random.sample(alpha,4)

mypswd=""

for c in chars:

mypswd+=c

ans = linsearch(pswdlist,mypswd)

if(ans == True):

amount+= 1

print("number of found passwords:",amount)

prob = 0

total = 0

tests = 0

count = 0

#Run until probability is 0.5

random.shuffle(alpha)

while(count < 1000):

count +=1

while(prob < 0.5):

newpswd = ""

chars = (random.sample(alpha,4))

for c in chars:

newpswd += c

pswdlist.append(newpswd)

#mypasswd

chars = random.sample(alpha,4)

mypswd=""

for c in chars:

mypswd+=c

ans = linsearch(pswdlist,mypswd)

if(ans == True):

total +=1

prob = 1 - total/len(pswdlist)

total += len(pswdlist)

print(total/1000)