**1. How much should I bet?**

Are you in for some betting? In this game there are some rules: a fair coin will be tossed until the first time it comes up heads. If this occurs on the jth toss you are paid 2 to power j dollars. You are sure to win at least 2 dollars so you should be willing to pay to play this game, but how much? Few people would pay as much as 10 dollars to play this game. See if you can decide, by simulation, a reasonable amount that you would be willing to pay, per game, if you will be allowed to make a large number of plays of the game. Does the amount that you would be willing to pay per game depend upon the number of plays that you will be allowed?

from random import randint  
total\_earnings = []  
N= 1000000  
def sums(arr):  
 sum=0  
 for elem in arr:  
 sum+=elem  
 return sum  
  
def winnings():  
 for i in range(N):  
 sum=0  
 flipped = 1   
 while(sum==0):  
 if (randint(1,2)==2):  
 sum = 2\*\*flipped  
 else:  
 flipped+=1  
 total\_earnings.append(sum)  
  
 total = sums(total\_earnings)  
 return total/N  
  
def main():  
 res = winnings()  
 print(res)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

It doesn’t depend on the number of plays that we are allowed to play, because the summation is a divergent.

### 2. I bet my life on this one

Are you still in the mood for betting? Now let’s play something more interesting and serious! Let's play some Russian roulette. Here's the deal. I have a revolver with a 6 slots barrel and only 2 bullets. Now I put the bullets into the revolver in **adjacent** slots, spin the barrel and hand you the gun. You point the gun to your head. You pull the trigger and ... Click! you're still alive. Congratulations but the game is not over yet. You have to pull the trigger one last time. Now you have two choices. 1 You spin the barrel afterwards you pull the trigger. 2 You pull the trigger without spinning. Luckily you have some time and a computer with you in your bag so you can simulate the current situation such that you can choose the correct choice.

Your assignment is to calculate the probability for both cases and after you've found the probability for the initial conditions you need also to find out what are the probabilities in case the bullets are not adjacent. Now you need to calculate the same probabilities but in case when you have 2 bullets and the gun has a 5 slots barrel. You have to present the result for 8 different outcomes. Good luck staying alive.

from random import randint  
  
N = 100000  
  
  
def empty(revolver):  
 for idx in range(len(revolver)):  
 revolver[idx]=0  
  
def load(revolver,adjacent):  
 empty(revolver)  
 if(adjacent):  
 rand = randint(0, len(revolver) - 1)  
 if rand != len(revolver) - 1:  
 revolver[rand] = revolver[rand + 1] = 1  
 else:  
 revolver[0] = revolver[-1] = 1  
 else:  
 bulletSlot = randint(0,len(revolver)-1)  
 bulletSlot2 = randint(0,len(revolver)-1)  
 revolver[bulletSlot]=1  
 while(bulletSlot2==bulletSlot or bulletSlot2+1==bulletSlot or bulletSlot2-1==bulletSlot):  
 bulletSlot2 = randint(0,len(revolver)-1)  
 revolver[bulletSlot]=revolver[bulletSlot2]=1  
  
def respin(revolver,adjacent):  
 alive=0  
 for \_ in range(N):  
 empty(revolver)  
 load(revolver,adjacent)  
 rand = randint(0,len(revolver)-1)  
 if(revolver[rand]==0):  
 alive+=1  
 return 100-((N - alive)/N \* 100)  
  
def stay(revolver,adjacent):  
 alive=0  
 for \_ in range(N):  
 load(revolver,adjacent)  
 rand = randint(0,len(revolver)-1)  
 while revolver[rand]!=0:  
 rand=randint(0,len(revolver)-1)  
 if rand-1 < 0:  
 if revolver[len(revolver)-1]==0:  
 alive+=1  
 elif revolver[rand-1]==0:  
 alive+=1  
 return 100-((N-alive)/N\*100)  
  
  
def main():  
 revolver = [0,0,0,0,0,0]  
 r = respin(revolver,True)  
 print(f'6 slots adjacent w re-spin chance of survival {r}%')  
 s = stay(revolver,True)  
 print(f'6 slots adjacent w no-spin chance of survival {s}%')  
 r = respin(revolver,False)  
 print(f'6 slots w respin chance of survival {r}%')  
 s = stay(revolver,False)  
 print(f'6 slots w no-spin chance of survival {s}%')  
 revolver = [0,0,0,0,0]  
 r = respin(revolver,True)  
 print(f'\n\n5 slots adjacent w respin chance of survival {r}%')  
 s = stay(revolver,True)  
 print(f'5 slots adjacent w no-spin chance of survival {s}%')  
 r = respin(revolver,False)  
 print(f'5 slots w respin chance of survival {r}%')  
 s = stay(revolver,False)  
 print(f'5 slots w no-spin chance of survival {s}%')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

### 3. Poker Hands

In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

* **High Card**: Highest value card.
* **One Pair**: Two cards of the same value.
* **Two Pairs**: Two different pairs.
* **Three of a Kind**: Three cards of the same value.
* **Straight**: All cards are consecutive values.
* **Flush**: All cards of the same suit.
* **Full House**: Three of a kind and a pair.
* **Four of a Kind**: Four cards of the same value.
* **Straight Flush**: All cards are consecutive values of the same suit.
* **Royal Flush**: Ten, Jack, Queen, King, Ace, in the same suit.

The cards are valued in the order:

2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then the highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hand** |  | **Player 1** |  | **Player 2** |  | **Winner** |
| **1** |  | 5H 5C 6S 7S KD  Pair of Fives |  | 2C 3S 8S 8D TD  Pair of Eights |  | Player 2 |
| **2** |  | 5D 8C 9S JS AC  Highest card Ace |  | 2C 5C 7D 8S QH  Highest card Queen |  | Player 1 |
| **3** |  | 2D 9C AS AH AC  Three Aces |  | 3D 6D 7D TD QD  Flush with Diamonds |  | Player 2 |
| **4** |  | 4D 6S 9H QH QC  Pair of Queens  Highest card Nine |  | 3D 6D 7H QD QS  Pair of Queens  Highest card Seven |  | Player 1 |
| **5** |  | 2H 2D 4C 4D 4S  Full House  With Three Fours |  | 3C 3D 3S 9S 9D  Full House  with Three Threes |  | Player 1 |

The file poker.txt(in the zip file from email), contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

def handConvert(cards):  
 card\_rank = {  
 '2': 0,  
 '3': 1,  
 '4': 2,  
 '5': 3,  
 '6': 4,  
 '7': 5,  
 '8': 6,  
 '9': 7,  
 'T': 8,  
 'J': 9,  
 'Q': 10,  
 'K': 11,  
 'A': 12  
 }  
 card\_suit ={  
 'C': 0,  
 'D': 1,  
 'H': 2,  
 'S': 3  
 }  
 result = []  
 for card in cards:  
 result.append([card\_rank[card[0]],card\_suit[card[1]]])  
 result.sort()  
 result.reverse()  
 return result  
  
  
def check\_royal(cards):  
 if(cards[0][0]==12):  
 if cards[0][0]-1 == cards[1][0] and cards[0][1] == cards[1][1]:  
 if cards[1][0]-1 == cards[2][0] and cards[1][1] == cards[2][1]:  
 if cards[2][0]-1 == cards[3][0] and cards[2][1] == cards[3][1]:  
 if cards[3][0]-1 == cards[4][0] and cards[3][1] == cards[4][1]:  
 return True  
 return False  
  
def check\_straightFlush(cards):  
 if cards[0][0]-1 == cards[1][0] and cards[0][1] == cards[1][1]:  
 if cards[1][0]-1 == cards[2][0] and cards[1][1] == cards[2][1]:  
 if cards[2][0]-1 == cards[3][0] and cards[2][1] == cards[3][1]:  
 if cards[3][0]-1 == cards[4][0] and cards[3][1] == cards[4][1]:  
 return cards[0][0]  
 return False   
  
def check\_fourOfKind(cards):  
 cardRanks = [cards[0][0],cards[1][0],cards[2][0],cards[3][0],cards[4][0]]  
 for cardRank in cardRanks:  
 if (cardRanks.count(cardRank)==4):  
 return cardRank  
 return False  
  
def check\_fullHouse(cards):  
 if(check\_threeOfKind(cards) and check\_onePair(cards)):  
 return True  
 return False  
  
def check\_flush(cards):  
 cardSuits = [cards[0][1],cards[1][1],cards[2][1],cards[3][1],cards[4][1]]  
 for cardSuit in cardSuits:  
 if(cardSuits.count(cardSuit)==5):  
 return cards[0][0]   
 return False  
  
def check\_straight(cards):  
 if cards[0][0]-1 == cards[1][0]:  
 if cards[1][0]-1 == cards[2][0]:  
 if cards[2][0]-1 == cards[3][0]:  
 if cards[3][0]-1 == cards[4][0]:  
 return cards[0][0]  
 return False   
  
def check\_threeOfKind(cards):  
 cardRanks = [cards[0][0],cards[1][0],cards[2][0],cards[3][0],cards[4][0]]  
 for cardRank in cardRanks:  
 if (cardRanks.count(cardRank)==3):  
 return cardRank  
 return False  
  
def check\_twoPair(cards):  
 cardRanks = [cards[0][0],cards[1][0],cards[2][0],cards[3][0],cards[4][0]]  
 pairRank =[]  
 pairs=0  
 for cardRank in cardRanks:  
 if (cardRanks.count(cardRank)==2):  
 pairRank.append(cardRank)  
 pairs+=1  
 cardRanks.remove(cardRank)  
 pairRank.sort()  
 if(pairs==2):  
 return pairRank[0]  
 return False  
  
def check\_onePair(cards):  
 cardRanks = [cards[0][0],cards[1][0],cards[2][0],cards[3][0],cards[4][0]]  
 for cardRank in cardRanks:  
 if (cardRanks.count(cardRank)==2):  
 return cardRank  
 return False  
   
def highCard(cards):  
 return cards[0][0]  
  
def main():  
 ties=0  
 p1=0  
 p2=0  
 with open(r'./poker.txt', 'r') as f:  
 for line in f:  
 cards\_p1 = []  
 cards\_p2 = []  
 line = line.strip().split(' ')  
  
 if line[-2:-1] == '\n':  
 line = line[:-2]  
   
 cards\_p1 = line[:5]  
 cards\_p2 = line[5:]  
  
 res1 = handConvert(cards\_p1)  
 res2 = handConvert(cards\_p2)  
  
 if check\_royal(res1)==True and check\_royal(res2)==True:  
 ties+=1  
 elif check\_royal(res1)==True and check\_royal(res2)==False:  
 p1+=1  
 elif check\_royal(res2)==True and check\_royal(res1)==False:  
 p2+=1  
  
 elif check\_straightFlush(res1) and check\_straightFlush(res2):  
 if check\_straight(res1)>check\_straightFlush(res2):  
 p1+=1  
 elif check\_straight(res2)>check\_straightFlush(res1):  
 p2+=1  
 elif check\_straightFlush(res1) and check\_straightFlush(res2)==False:  
 p1+=1  
 elif check\_straightFlush(res1)==False and check\_straightFlush(res2):  
 p2+=1  
  
  
 elif check\_fourOfKind(res1) and check\_fourOfKind(res2):  
 if check\_fourOfKind(res1)>check\_fourOfKind(res2):  
 p1+=1  
 elif check\_fourOfKind(res2)>check\_fourOfKind(res1):  
 p2+=1  
 elif check\_fourOfKind(res1) and check\_fourOfKind(res2)==False:  
 p1+=1  
 elif check\_fourOfKind(res1)==False and check\_fourOfKind(res2):  
 p2+=1  
  
 elif check\_fullHouse(res1) and check\_fullHouse(res2):  
 if check\_threeOfKind(res1)>check\_threeOfKind(res2):  
 p1+=1  
 elif check\_threeOfKind(res2)>check\_threeOfKind(res1):  
 p2+=1  
 if check\_onePair(res1)>check\_onePair(res2):  
 p1+=1  
 elif check\_onePair(res1)<check\_onePair(res2):  
 p2+=1  
  
 elif check\_fullHouse(res1) and check\_fullHouse(res2)==False:  
 p1+=1  
 elif check\_fullHouse(res1)==False and check\_fullHouse(res2):  
 p2+=1   
  
 elif check\_flush(res1) and check\_flush(res2):  
 if check\_flush(res1)>check\_flush(res2):  
 p1+=1  
 elif check\_flush(res2)>check\_flush(res1):  
 p2+=1  
   
 elif check\_flush(res1) and check\_flush(res2)==False:  
 p1+=1  
 elif check\_flush(res1)==False and check\_flush(res2):  
 p2+=1   
  
 elif check\_straight(res1) and check\_straight(res2):  
 if check\_straight(res1)>check\_straight(res2):  
 p1+=1  
 elif check\_straight(res2)>check\_straight(res1):  
 p2+=1  
 elif check\_straight(res1) and check\_straight(res2)==False:  
 p1+=1  
 elif check\_straight(res1)==False and check\_straight(res2):  
 p2+=1   
  
 elif check\_threeOfKind(res1) and check\_threeOfKind(res2):  
 if check\_threeOfKind(res1)>check\_threeOfKind(res2):  
 p1+=1  
 elif check\_threeOfKind(res2)>check\_threeOfKind(res1):  
 p2+=1  
   
 elif check\_threeOfKind(res1) and check\_threeOfKind(res2)==False:  
 p1+=1  
 elif check\_threeOfKind(res1)==False and check\_threeOfKind(res2):  
 p2+=1   
  
 elif check\_twoPair(res1) and check\_twoPair(res2):  
 if check\_twoPair(res1)>check\_twoPair(res2):  
 p1+=1  
 elif check\_twoPair(res2)>check\_twoPair(res1):  
 p2+=1  
   
 elif check\_twoPair(res1) and check\_twoPair(res2)==False:  
 p1+=1  
 elif check\_twoPair(res1)==False and check\_twoPair(res2):  
 p2+=1   
  
 elif check\_onePair(res1) and check\_onePair(res2):  
 if check\_onePair(res1)>check\_onePair(res2):  
 p1+=1  
 elif check\_onePair(res2)>check\_onePair(res1):  
 p2+=1  
 elif check\_onePair(res1) and check\_onePair(res2)==False:  
 p1+=1  
 elif check\_onePair(res1)==False and check\_onePair(res2):  
 p2+=1  
 elif highCard(res1)>highCard(res2):  
 p1+=1  
 elif highCard(res2)>highCard(res1):  
 p2+=1  
  
 print(f"Player 1 : {p1} wins")  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

How many hands does Player 1 win?



### 4. 21 Game

Ivan plays the following game, loosely based on the card game **"21"**.

Ivan starts with 0 points and draws numbers while he has less than **k** points. During each draw, he gains an integer number of points randomly from the range [1,**maxPts**], where **maxPts** is an integer. Each draw is independent and the outcomes have equal probabilities.

Ivan stops drawing numbers when he gets **k or more points**.

Return the probability that Ivan has **n** or fewer points.

**Example 1:**

**Input:** n = 10, k = 1, maxPts = 10

**Output:** 1.00000

**Explanation:** Ivan gets a single card, then stops.

**Example 2:**

**Input:** n = 6, k = 1, maxPts = 10

**Output:** 0.60000

**Explanation:** Ivan gets a single card, then stops.

In 6 out of 10 possibilities, he is at or below 6 points.

**Example 3:**

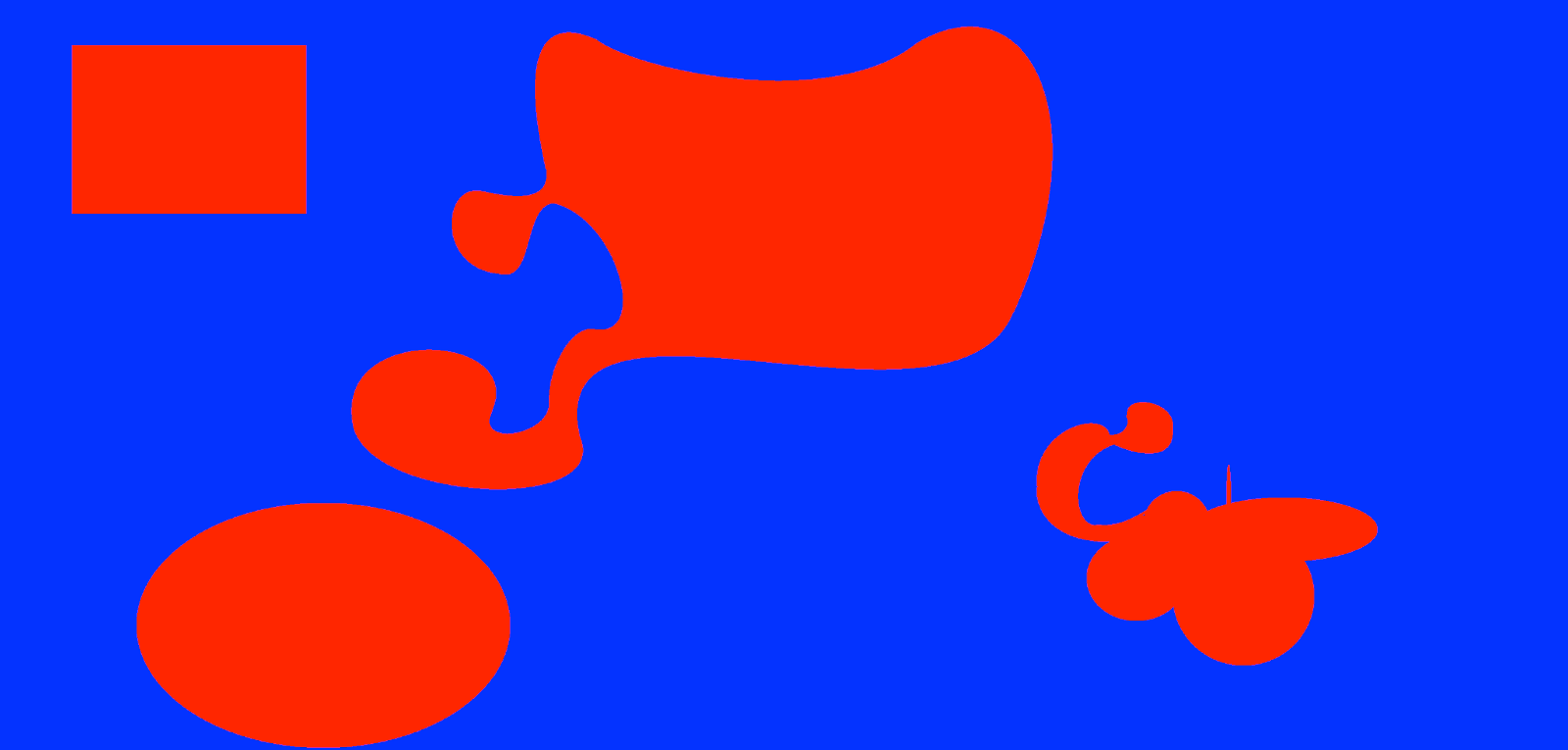
**Input:** n = 21, k = 17, maxPts = 10

**Output:** 0.73278

from random import randint  
def game(n,k,maxpts):  
 N = 100000  
 res=0  
 for \_ in range(N):  
 start=0  
 while(start<k):  
 rand = randint(1,maxpts)  
 start += rand  
 if(start<=n):  
 res+=1  
 return (res/N)  
   
def main():  
 n = 21  
 k = 17  
 maxpts = 10  
  
 res = game(n,k,maxpts)  
 print(res)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Simple implementation of the condition with 100.000 test cases and returning the average

### BONUS: PROBLEM



This morning the CIA faxed UTM the following map scan(image is also in the zip file from the email). The total captured surface is around 42 square miles (duh... this imperial metric). With red is marked the land that is mined by the guerrilla forces and at the moment they need to evaluate the logistics required to defuse the deadly mines. But they do not have enough computing resources to compute the red area from the map. Which is why they are playing their trump card, the brilliant engineers from FAF.

The stakes are very high, and lots of innocent lives can be spared. So please compute the mined area a.s.a.p., using Monte Carlo method.

**Tip:**

The recommended python library for image processing is **pillow** (not the one you use to sleep).

from PIL import Image  
from random import randint  
def main():  
 image = Image.open('danger\_zone.png')  
 red = 0  
 blue = 0  
 redArea = 0  
 imrgb = image.convert("RGB")  
 #for px in imrgb.getdata():  
 # if px == (255,0,0):  
 # red+=1  
 # else:  
 # blue+=1  
 # redArea = red/(red+blue)\*42  
 n = 10000000  
 for \_ in range(n):  
 x = randint(0,image.width-1)  
 y = randint(0,image.height-1)  
 coor = x,y  
 if (imrgb.getpixel( coor ) == (255,0,0)):  
 red+=1  
 else:  
 blue+=1  
 redArea = (red/(red+blue)\*42)  
 print(redArea)  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

By using Monte Carlo’s method we are picking random pixels are x and y position from the image and get its colour(red or blue) and at the end of our counting we are returning the area of the red pixels.

Monte Carlo’s method implies that the more random cases are being used the more closer we get to the actual answer we are looking for. 1 million test cases got me close to the actual answer(the actual answer can be computed by using the commented code).