**Assignment 1: Simulations (Given: 17 Jan 2023, Due: 31 Jan 2023 2pm)**

**General instructions**

- Solutions are to be typed in the `.ipynb` file provided and uploaded in the lab course page in Moodle before the due date.
- Your code should be well commented and should be compatible with python3.
- For this assignment, you are allowed to import the libraries `random` and `mathplotlib` of python3. No other libraries may be imported.
- For questions involving constructing plots, sample outputs may be provided. Your answers need not exactly match them. However if your plots are significantly deviating from what is given, it is possibily an indication that your code is doing something incorrect.

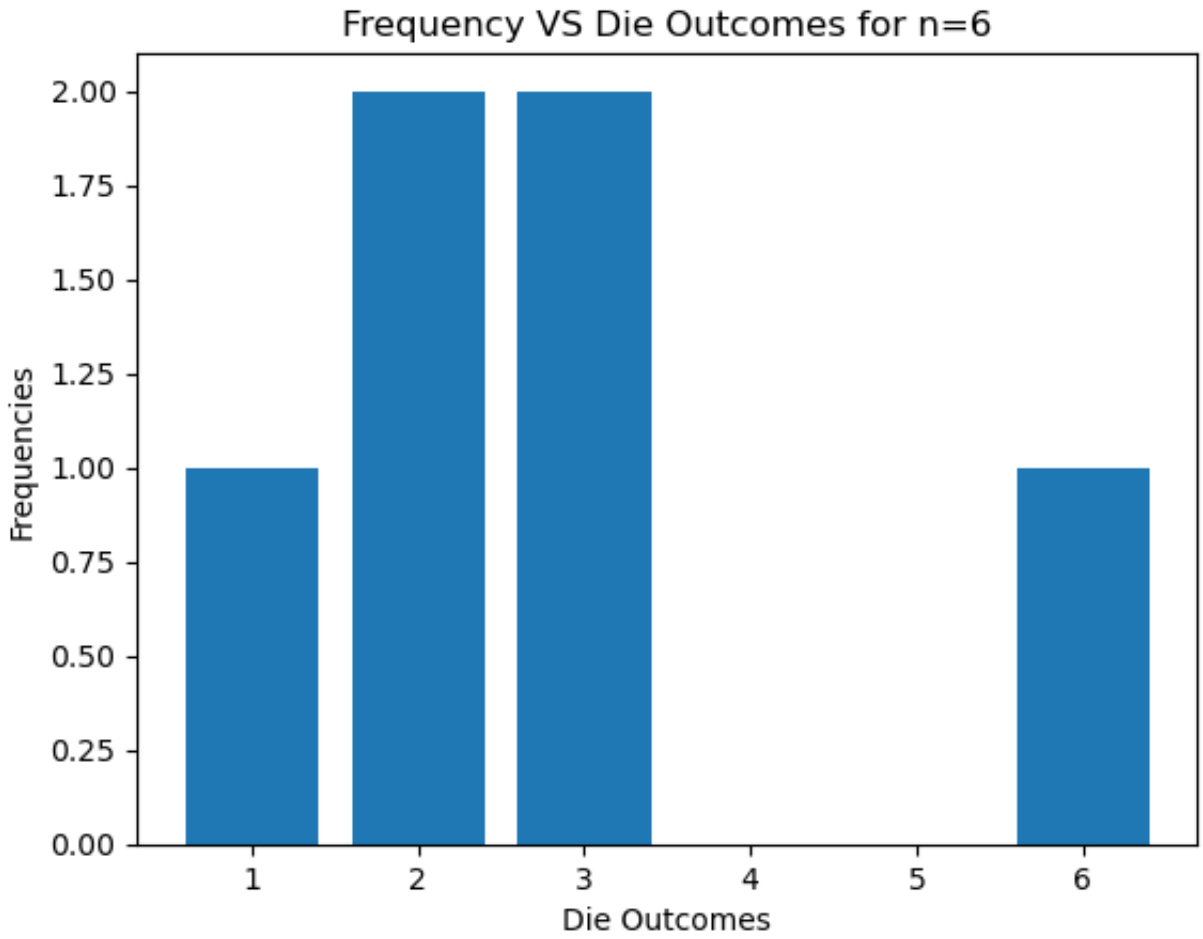
# Law of Large Numbers

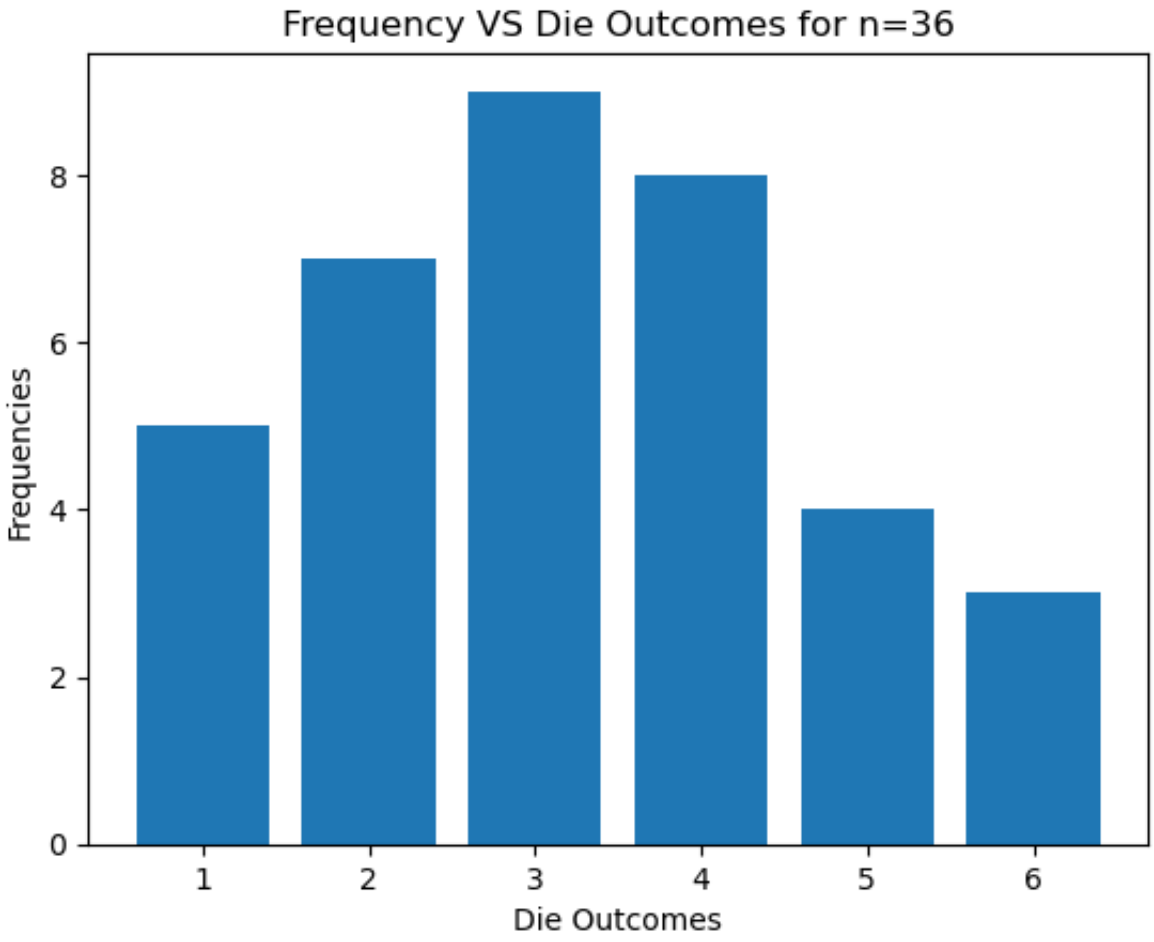
(a) Write a function `simulateDice(n)` that rolls a six-sided die $n$ times and returns the frequencies of the results, i.e., the number of times 1 appears, the no. of times 2 appears, etc.

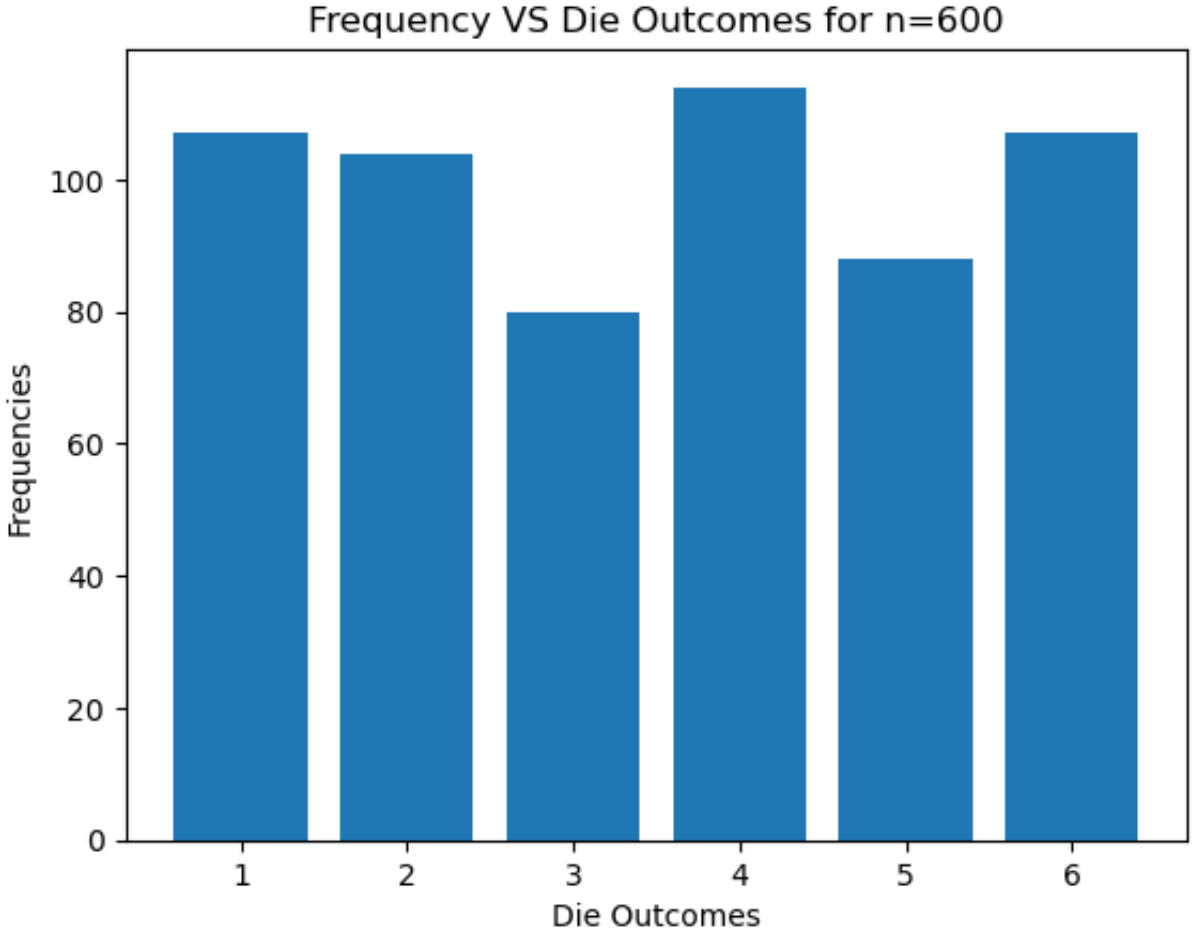
In [ ]:
```
import random

import matplotlib.pyplot as plt


def simulateDice(n):

    freq = [0 for i in range(6)]
#     using list comprehension to create a list of 6 elements initialized
    for i in range(n):
        obtainedNum = random.randint(1,6)
#         simulate a die throw
        freq[obtainedNum - 1]+=1
#     incrementing the frequency of the obtained number by 1
    return freq
```

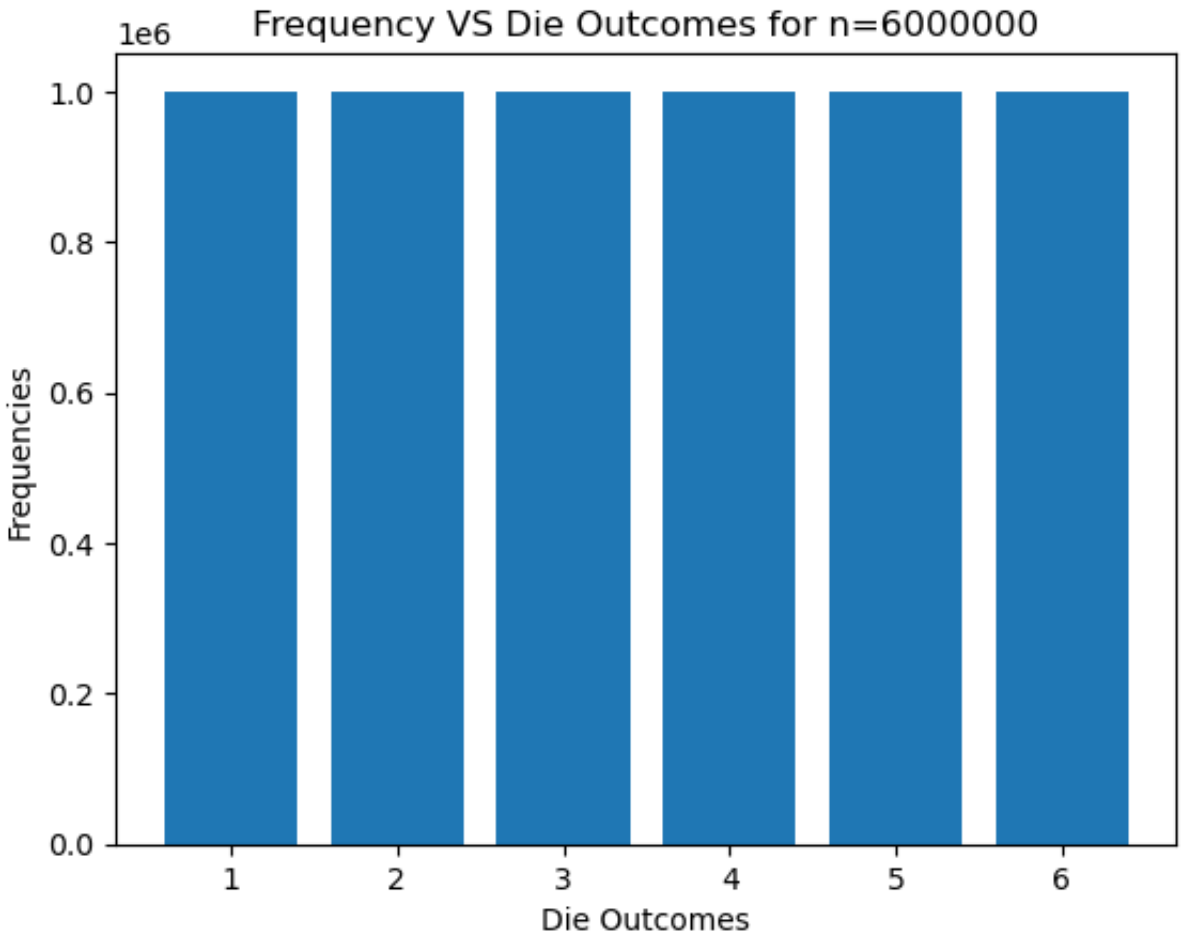
(b) For each $n \in \{6, 6^2, 60, 600, 6 \times 10^6\}$, run `simulateDice(n)` and plot a bar chart with outcomes of die rolls against the frequencies. A sample output looks as follows.

In [ ]:
```
listForPlotting = [6,36,600,int(6e6)]

for n in listForPlotting:
    yAxis = simulateDice(n)
    #yAxis contains the list of frequencies
    xAxis = [i+1 for i in range(6)]
    #xAxis contains the list [1,2,3,4,5,6]
    plt.bar(xAxis,yAxis)
    #to plot the bar graph
    plt.xlabel('Die Outcomes')
    plt.ylabel('Frequencies')
    #xLabel and yLabel assigned
    plt.title(f'Frequency VS Die Outcomes for n={n}')
    plt.show()
```



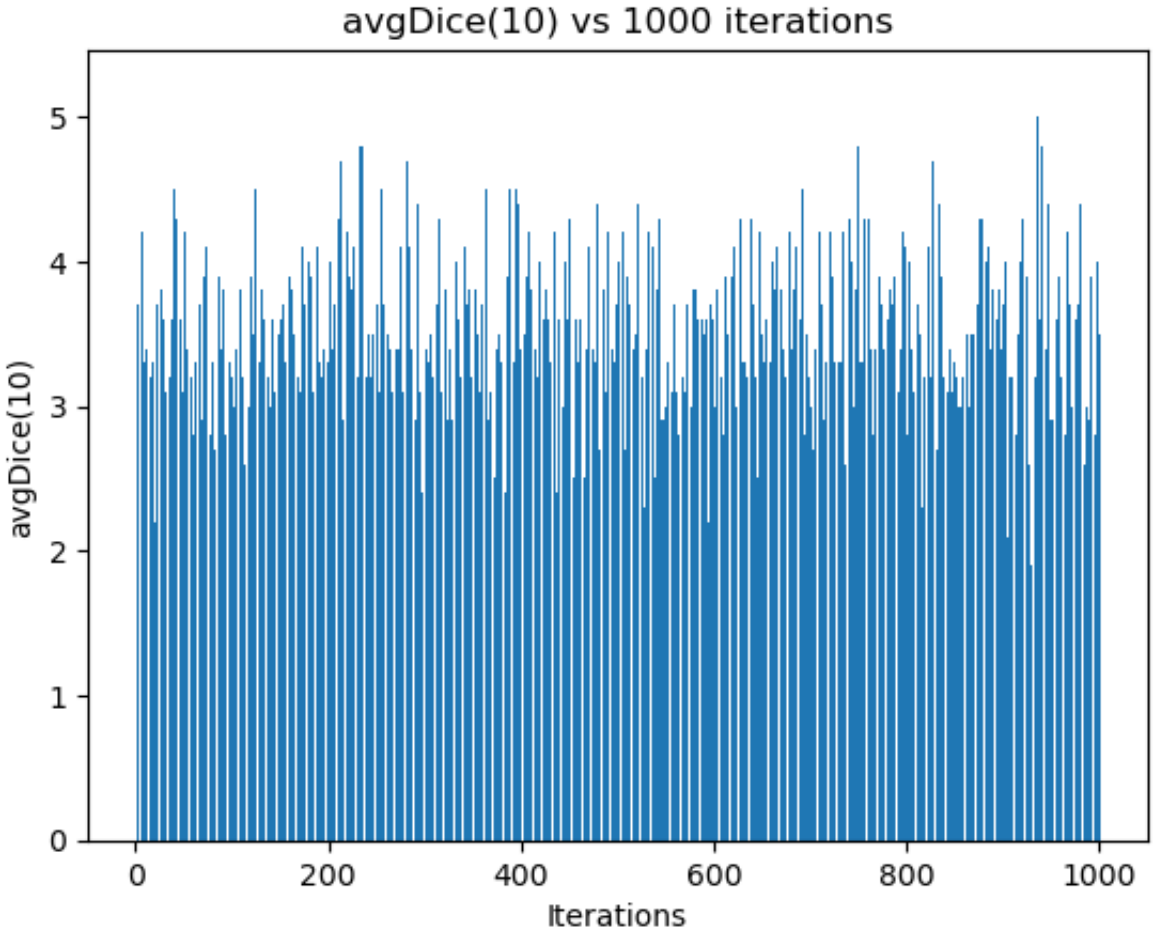

Frequency VS Die Outcomes for n=36
Frequencies
0
2
4
6
8
1
2
3
4
5
6
Die Outcomes


Frequency VS Die Outcomes for n=600
Frequencies
0
20
40
60
80
100
1
2
3
4
5
6
Die Outcomes



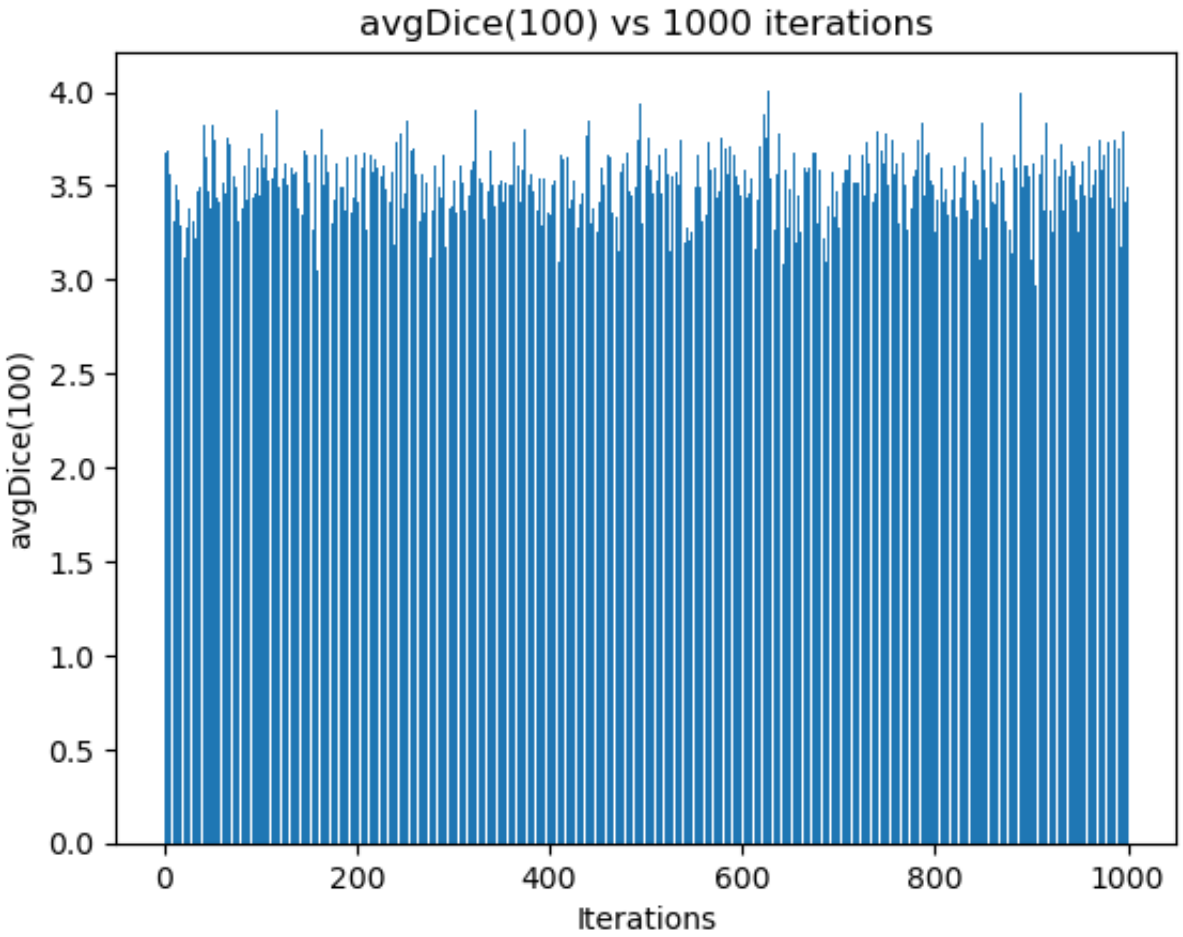

(c) Write a function `avgDice(n)` that calls `simulateDice(n)` and compute the average of the $n$ outcomes. Run avgDice(n) 1000 times and plot a bar chart with iterations (i.e., 1 to 1000) against the average values. A sample output looks for $n = 10$ as follows.

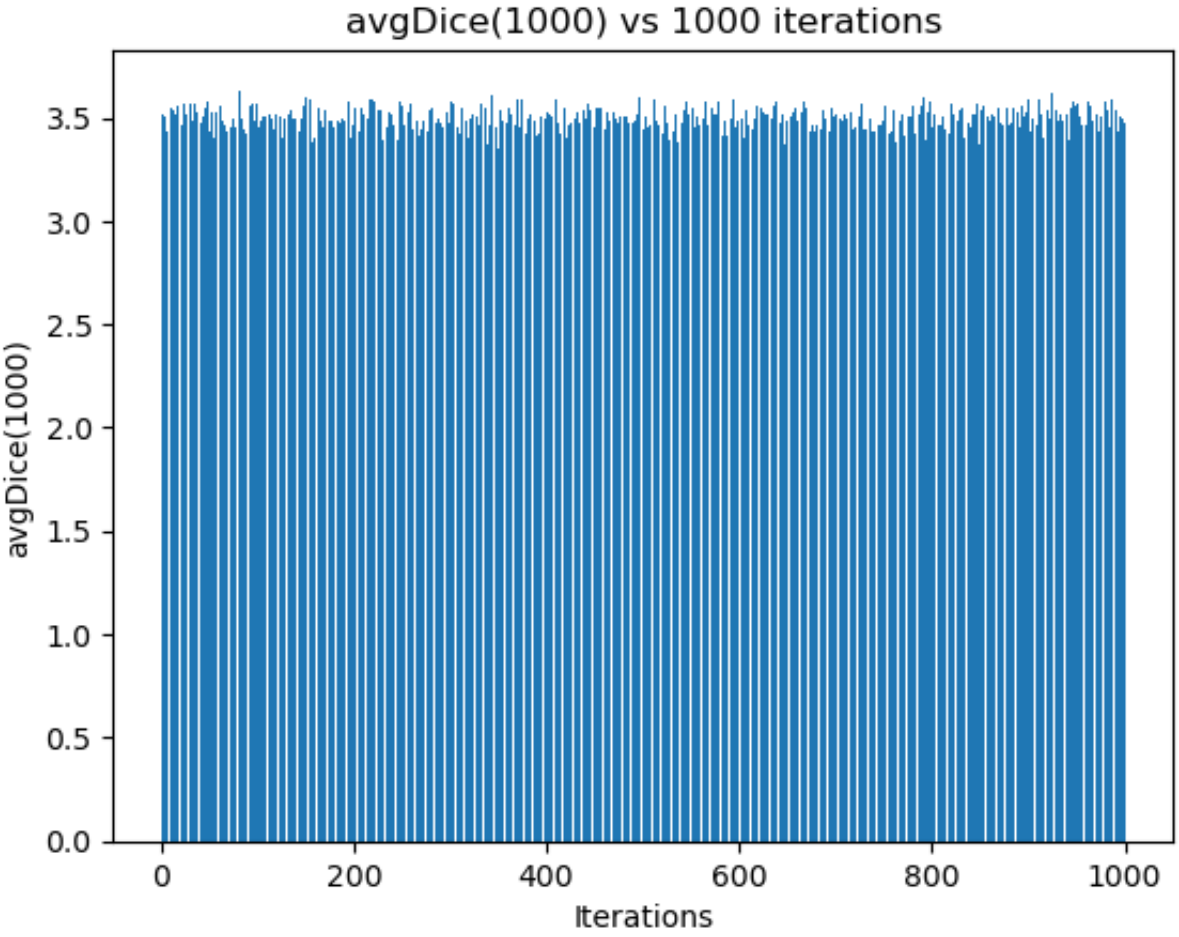
In [ ]:
```
def avgDice(n):
    listForComputingAvg = simulateDice(n)
    sumList = 0
    for i in range(6):
        sumList += listForComputingAvg[i] * (i+1)
#     print(n)
#     print(sum(listForComputingAvg))
#     return sumList/sum(listForComputingAvg)
    return sumList / n

#for n=10

yAxis = [avgDice(10) for i in range(1000)]
#calling avgDice(10) for 1000 times
xAxis = [i+1 for i in range(1000)]
#iteration count
plt.bar(xAxis,yAxis)
#to plot bar chart
plt.xlabel('Iterations')
plt.ylabel('avgDice(10)')
# xLabel and yLabel assigned
plt.title('avgDice(10) vs 1000 iterations')
#plot title
plt.show()
```

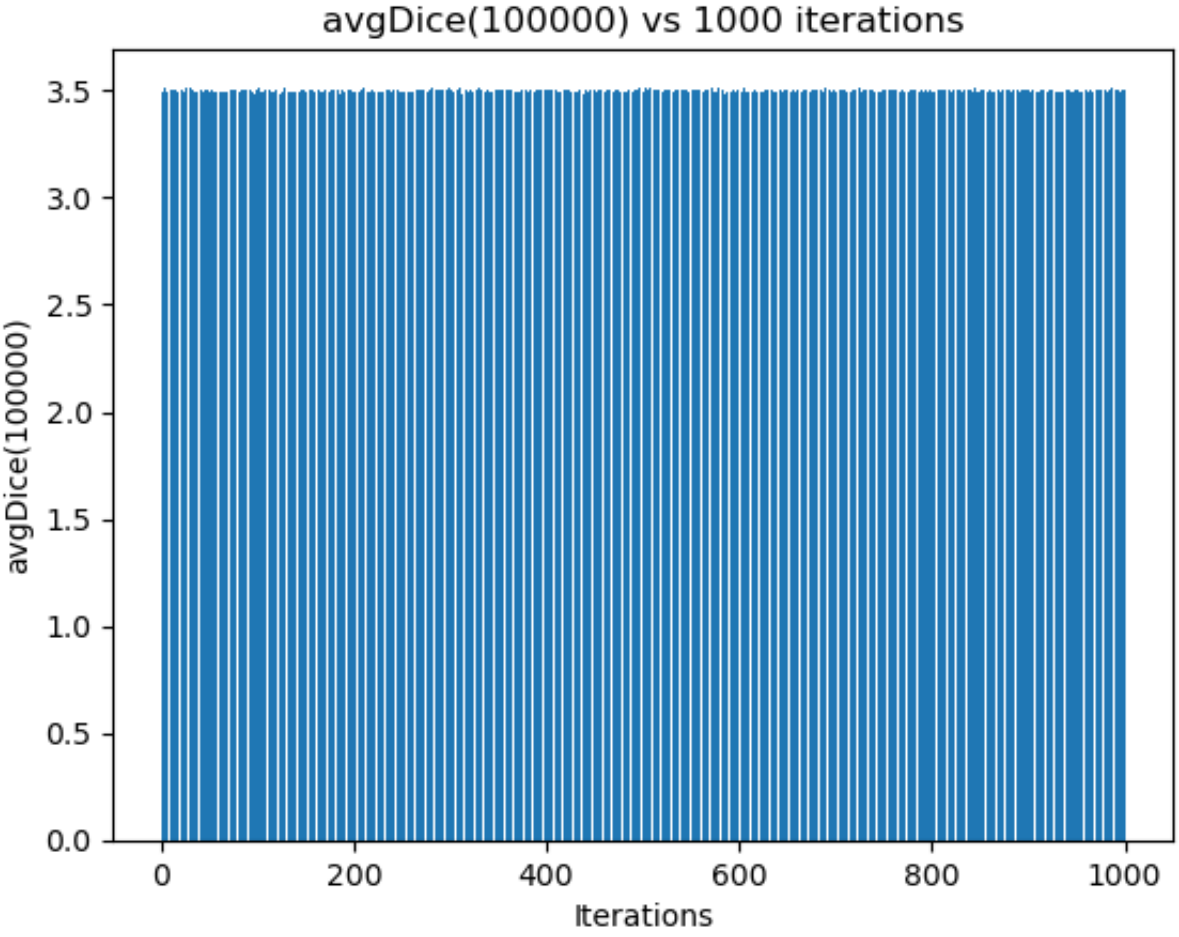


(d) Repeat part (c) for each $n \in \{10^2, 10^3, 10^5\}$. How does the chart change with $n$ ?

In [ ]:
```
lst = [100,1000,int(1e5)]
for n in lst:
    yAxis = [avgDice(n) for i in range(1000)]
    #calling avgDice(n) for 1000 times
    xAxis = [i+1 for i in range(1000)]
    #iteration count
    plt.bar(xAxis,yAxis)
    #to plot bar chart
    plt.xlabel('Iterations')
    plt.ylabel(f'avgDice({n})')
    # xLabel and yLabel assigned
    plt.title(f'avgDice({n}) vs 1000 iterations')
    #plot title
    plt.show()
```




avgDice(1000) vs 1000 iterations
avgDice(1000)
Iterations
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
200
400
600
800
1000


avgDice(100000) vs 1000 iterations
avgDice(100000)
Iterations
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
200
400
600
800
1000

**Observation** : As *n* increases, the probability of each die face approaches 1/6, and the average approches ***(1+2+3+4+5+6) / 6*** which is ***3.5***.

# Birthday Paradox

(a) Write a function `simulateBday(n)` that generates $n$ random birthdays (i.e., $n$ random integers between 1 and 366) and determines whether there is a pair of same birthdays or not.

In [ ]:
```
def simulateBday(n):
    randomBdays = [random.randint(1,366) for i in range(n)]
#     the above is a list of n random birthdays
    count = 0
    for i in range(1,367):
        if(randomBdays.count(i)>=2):
#             if there exists a birthday whose count is greater than 2, t
            return True
#     if the whole for loop finishes without returning true, then it mean
    return False
```

(b) Run `simulateBday(n)` 100 times and determine the number $x$ of times that the $n$ birthdays have an equal pair. Treat $x/100$ as the probability of two same birthdays among $n$ random birthdays.

In [ ]:
```
def probSameBday(n):
    x = 0
#     x is initially zero
    for i in range(100):
#         repeating 100 times
        if(simulateBday(n)):
#             if simulateBday(n) returns True, then it means that there e
#             increment x
            x+=1
    return x/100
```

(c) Run `simulateBday(n)` for $n \in [100]$ where $[k]$ denotes the set $\{1, 2, \ldots, k\}$.

In [ ]:
```
# running simulateBday(n)
for i in range(1,101):
    if(simulateBday(i)==True):
        print(f'For n={i} ("YES")')
    else:
        print(f'For n={i} ("NO")')


# print('The probability that there exist an equal birthday pair for')
# for i in range(1,101):
#     ch = ''
#     if i>1:
#         ch='s'

#     probability = probSameBday(i)
# #     printing the probabilities
#     print(f'{i} birthday{ch} is {probability}')
```

```
For n=1 ("NO")
For n=2 ("NO")
For n=3 ("NO")
For n=4 ("NO")
For n=5 ("NO")
For n=6 ("NO")
For n=7 ("NO")
For n=8 ("NO")
For n=9 ("NO")
For n=10 ("NO")
For n=11 ("NO")
For n=12 ("YES")
For n=13 ("YES")
For n=14 ("NO")
For n=15 ("NO")
For n=16 ("NO")
For n=17 ("NO")
For n=18 ("YES")
For n=19 ("NO")
For n=20 ("YES")
For n=21 ("NO")
For n=22 ("YES")
For n=23 ("NO")
For n=24 ("NO")
For n=25 ("NO")
For n=26 ("NO")
For n=27 ("YES")
For n=28 ("YES")
For n=29 ("NO")
For n=30 ("NO")
For n=31 ("YES")
For n=32 ("YES")
For n=33 ("YES")
For n=34 ("NO")
```

```
For n=35 ("NO")
For n=36 ("NO")
For n=37 ("YES")
For n=38 ("YES")
For n=39 ("YES")
For n=40 ("YES")
For n=41 ("YES")
For n=42 ("YES")
For n=43 ("YES")
For n=44 ("YES")
For n=45 ("YES")
For n=46 ("YES")
For n=47 ("YES")
For n=48 ("YES")
For n=49 ("YES")
For n=50 ("YES")
For n=51 ("YES")
For n=52 ("YES")
For n=53 ("YES")
For n=54 ("YES")
For n=55 ("YES")
For n=56 ("YES")
For n=57 ("YES")
For n=58 ("YES")
For n=59 ("YES")
For n=60 ("YES")
For n=61 ("YES")
For n=62 ("YES")
For n=63 ("YES")
For n=64 ("YES")
For n=65 ("YES")
For n=66 ("YES")
For n=67 ("YES")
For n=68 ("YES")
For n=69 ("YES")
For n=70 ("YES")
For n=71 ("YES")
For n=72 ("YES")
For n=73 ("YES")
For n=74 ("YES")
For n=75 ("YES")
For n=76 ("YES")
For n=77 ("YES")
For n=78 ("YES")
For n=79 ("YES")
For n=80 ("YES")
For n=81 ("YES")
For n=82 ("YES")
For n=83 ("YES")
For n=84 ("YES")
For n=85 ("YES")
For n=86 ("YES")
For n=87 ("YES")
For n=88 ("YES")
For n=89 ("YES")
For n=90 ("YES")
For n=91 ("YES")
```

```
For n=92 ("YES")
For n=93 ("YES")
For n=94 ("YES")
For n=95 ("YES")
For n=96 ("YES")
For n=97 ("YES")
For n=98 ("YES")
For n=99 ("YES")
For n=100 ("YES")
```

(d) What is the minimum $n$ that guarantees two same birthdays? Identify a range of values of $n$ that makes the probability of two same birthdays greater than $0.5$.

In [ ]:
```
minN = 0
for i in range(100):
    if(int(probSameBday(i+1)) == 1):
#         to check if there is an equal birthday pair
        minN = i+1
        break
print(f'The minimum number of attempts to guarantee that there exist an s

# minNhalf = 0
# for i in range(100):
#     if probSameBday(i+1) > 0.5:
#         minNhalf = i+1
#         break
# print(f'The minimum number of attempts that makes probability > 0.5 is


minList = []
# initializing minList to an empty list


for i in range(100):
    for j in range(1,101):
        x = probSameBday(j)
#         x has the probability when n is j
        if(x > 0.5 and j not in minList):
#         if probability is greater than 0.5 and the corresponding j, if
#             , we should append it to the minList to find the range
            minList.append(j)
            break
minList.sort()
# sorting the minList
print('The range of values of n for which probability is greater than 0.5
print(minList)
```

```
The minimum number of attempts to guarantee that there exist an same birt
hday pair is 54
The range of values of n for which probability is greater than 0.5 is
[20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 5
6, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74
, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100]
```

# Monty Hall Game

You are on a game show and given the choice of picking one of the three boxes ($X$, $Y$, $Z$). Two of the boxes are empty and one contains a gift. You pick a box, say $X$, and the host (who knows what is inside each of the boxes) opens another box, say $Y$, which is empty. You now have the option of retaining your choice ($X$) or switching it (to $Z$). Is it to your advantage to switch your choice in order to get the gift? This advantage is defined as the fraction of times switching the choice leads to a win when the game is played 1000 times.

In [ ]:
```
# def simulateMontyHall():
#     giftIndex = random.randint(1,3)
# #     to randomly fix an index containing the gift

#     isGift = [0,0,0]
#     #     isGift is a list whose elements are 1 if that box contains gi
#     isGift[giftIndex - 1] = 1
# #     we have chosen 1st box, and the host opened the second box, we ha
# # the third box contains one
# # the host opens the box which is empty
#     if(isGift[1] == 1):
# #         if second box has gift, the host opens the third box
# #         and we have to find whether it is to our advantage to switch
#         return 1
#     elif(isGift[2] == 1):
#         return 1
#     else:
#         return 0

def simulateMontyHall():
    isGift = [0 for i in range(3)]
    giftIndex = random.randint(0,2)
#     index where the gift is present
    isGift[giftIndex] = 1
#     assigning element of isGift as 1 at the giftIndex

    firstChoice = random.randint(0,2)
#     first choice of the player
#     nonGiftIndex = [index for i in range(3) if  ]
    montyChoices = []
#     the choices of doors which can be opened by monty
    for i in range(3):
        if(i != firstChoice):
```

```
#             monty will not open the door chosen by player
            montyChoices.append(i)
    openedDoor = 0
    if(montyChoices[0] == giftIndex):
#         monty won't open the door containing the gift
#         monty opens the other door
        openedDoor = montyChoices[1]
    else:
        openedDoor = montyChoices[0]

    switchedDoorIndex = 0
    for i in range(3):
        if i==firstChoice or i==openedDoor:
            continue
        else:
            switchedDoorIndex = i
#             assuming we are switching to the door unopened by monty and


#     print(giftIndex,firstChoice,openedDoor)
#     print(switchedDoorIndex)
    if(isGift[switchedDoorIndex] == 1):
#         if the switched door has a gift, return True
        return True
    else:
        return False


# gameList is list containing elements such that the element is 1 if game
gameList = [simulateMontyHall() for i in range(1000)]
countWins = 0
for i in range(len(gameList)):
    countWins += gameList[i]
print(f'Advantage when the game is played 1000 times is {countWins/len(ga
```

```
Advantage when the game is played 1000 times is 0.655
```

**Observation**: It is to our advantage to switch our choice in order to get the gift

# Game of Dice

Simulate the following dice game.

1. Roll two six-sided dice.
    - If the sum is 7 or 11 on the first roll, you win.
    - If the sum is 2, 3 or 12 on the first roll, you lose.
    - If the sum is 4, 5, 6, 8, 9 or 10 on the first roll, that sum becomes your point.
2. Continue rolling the dice until the sum is 7 (game lost) or equal to your point (game won).

If $n$ games are played and $q$ of these games are won, then the chance of winning is $q/n$. Sample simulation of the game for $n = 10$ is as follows. What is the chance of winning at this game?

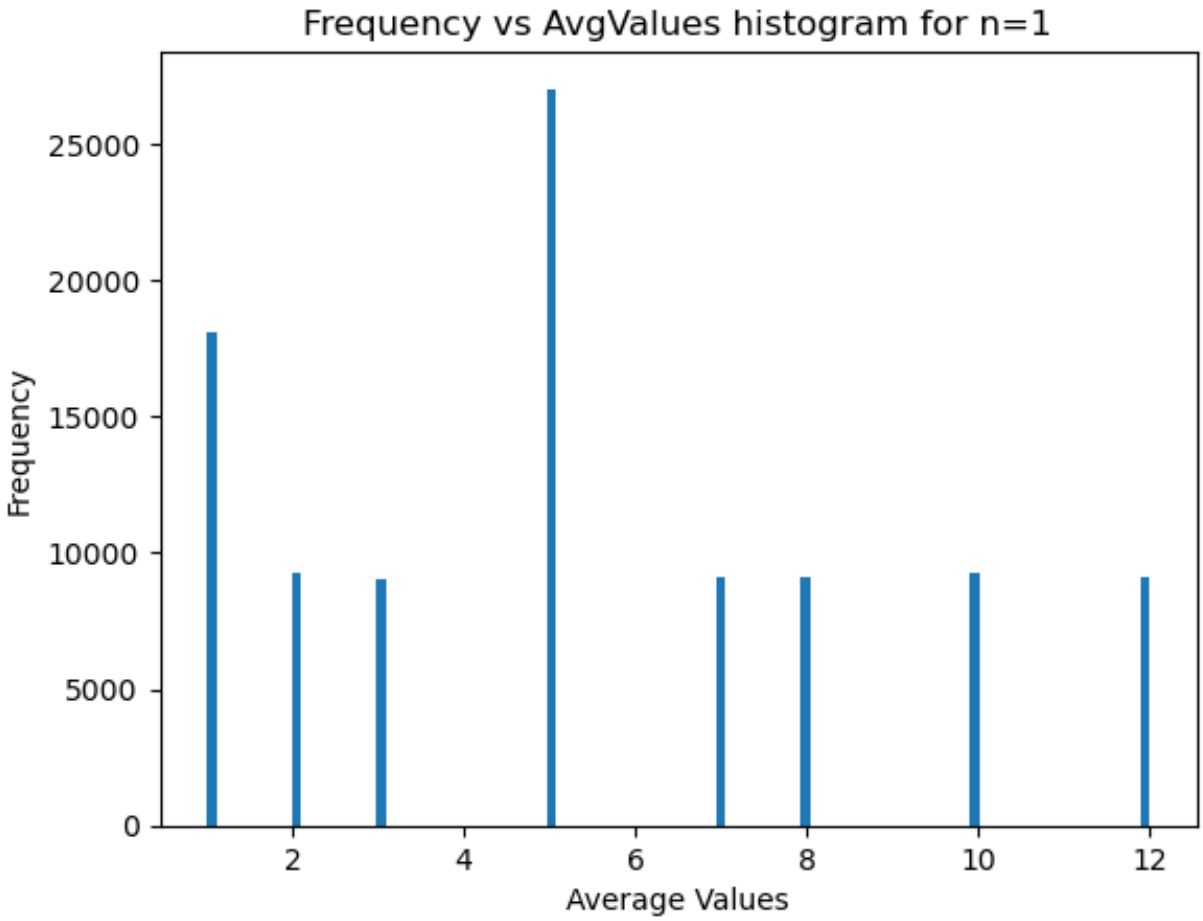
In [ ]:
```
# the below function simulates die throw
def throw():
    return random.randint(1,6)



def simulateGameOfDice(n):
    q = 0
    for i in range(n):
        print('----------')

        x = throw()
        y = throw()
        print(f'Player rolled {x} + {y} = {x+y}')
        if(x+y) in [2,3,12]:
#             if sum is one of the three numbers, then the player loses
            print('Player loses')
            continue
        elif x+y in [7,11]:
#             if the sum on the first roll is 7 or 11
            print('Player wins')
            q+=1
            continue

        else:
            point = x+y
            print(f'Point is {point}')
            x = throw()
            y = throw()
            while(x+y != 7 and x+y!=point):
#                 as long as the sum is not 7 or the point
                x = throw()
                y = throw()
                print(f'Player rolled {x} + {y} = {x+y}')
            if(x+y == 7):
                print('Player loses')
                continue
            else:
                print('Player wins')
                q+=1
                continue
    return q







n = 10
q = simulateGameOfDice(n)
print('----------')
print(f'Total number of wins is {q}')
# print(f'Win Fraction is {q/n}')
```

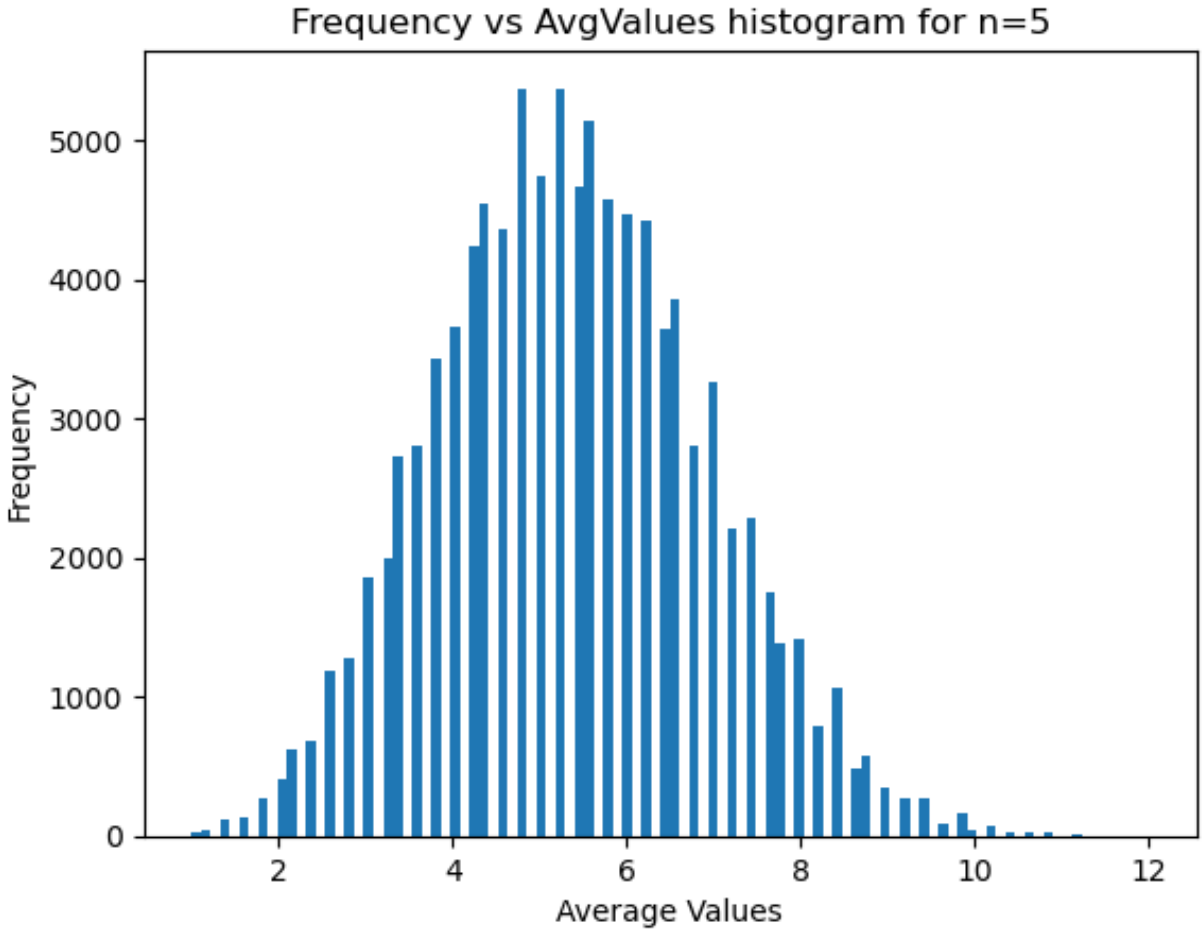
```
----------
Player rolled 5 + 6 = 11
Player wins
----------
Player rolled 6 + 6 = 12
Player loses
----------
Player rolled 2 + 4 = 6
Point is 6
Player loses
----------
Player rolled 2 + 6 = 8
Point is 8
Player wins
----------
Player rolled 1 + 4 = 5
Point is 5
Player loses
----------
Player rolled 2 + 1 = 3
Player loses
----------
Player rolled 6 + 1 = 7
Player wins
----------
Player rolled 5 + 2 = 7
Player wins
----------
Player rolled 4 + 4 = 8
Point is 8
Player rolled 3 + 5 = 8
Player wins
----------
Player rolled 4 + 5 = 9
Point is 9
Player rolled 4 + 3 = 7
Player loses
----------
Total number of wins is 5
```

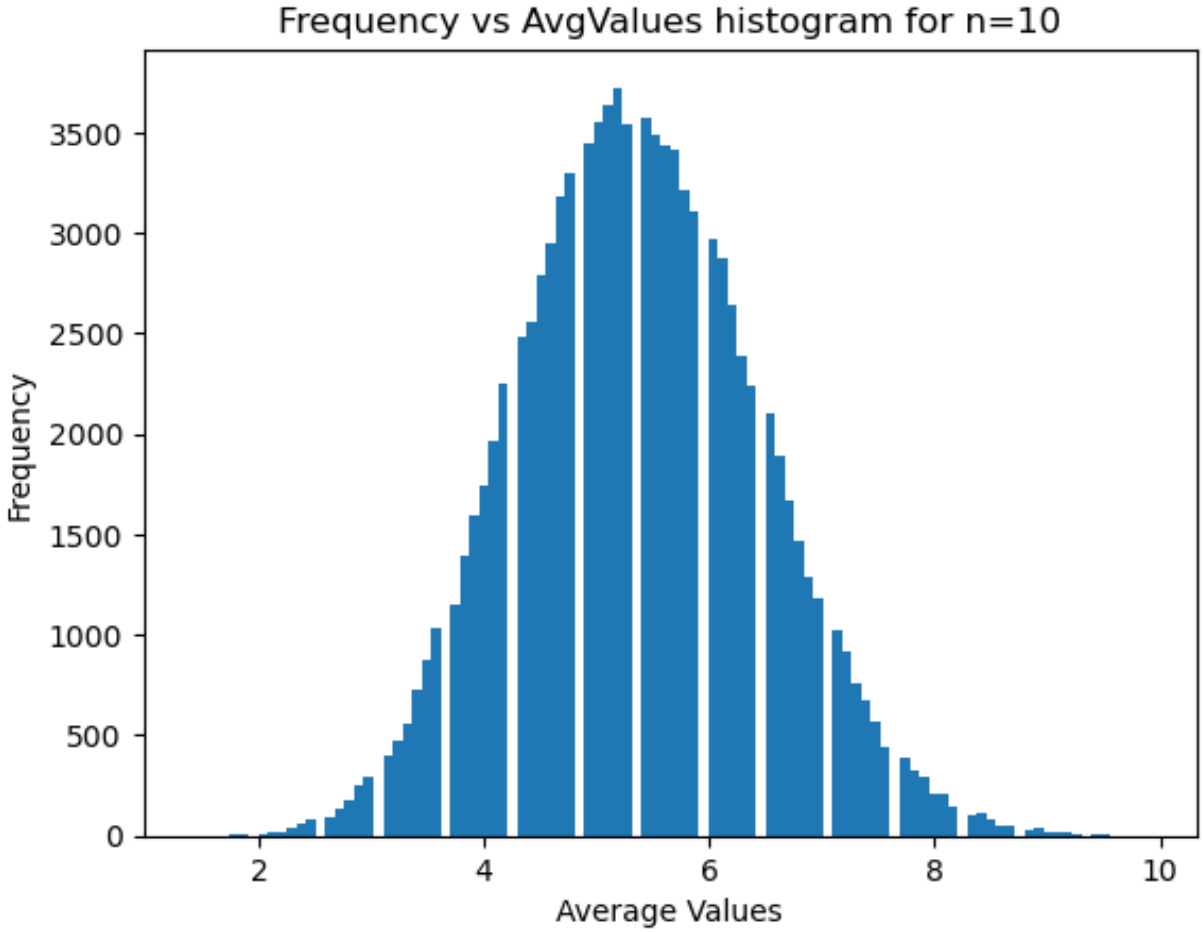
# Central Limit Theorem

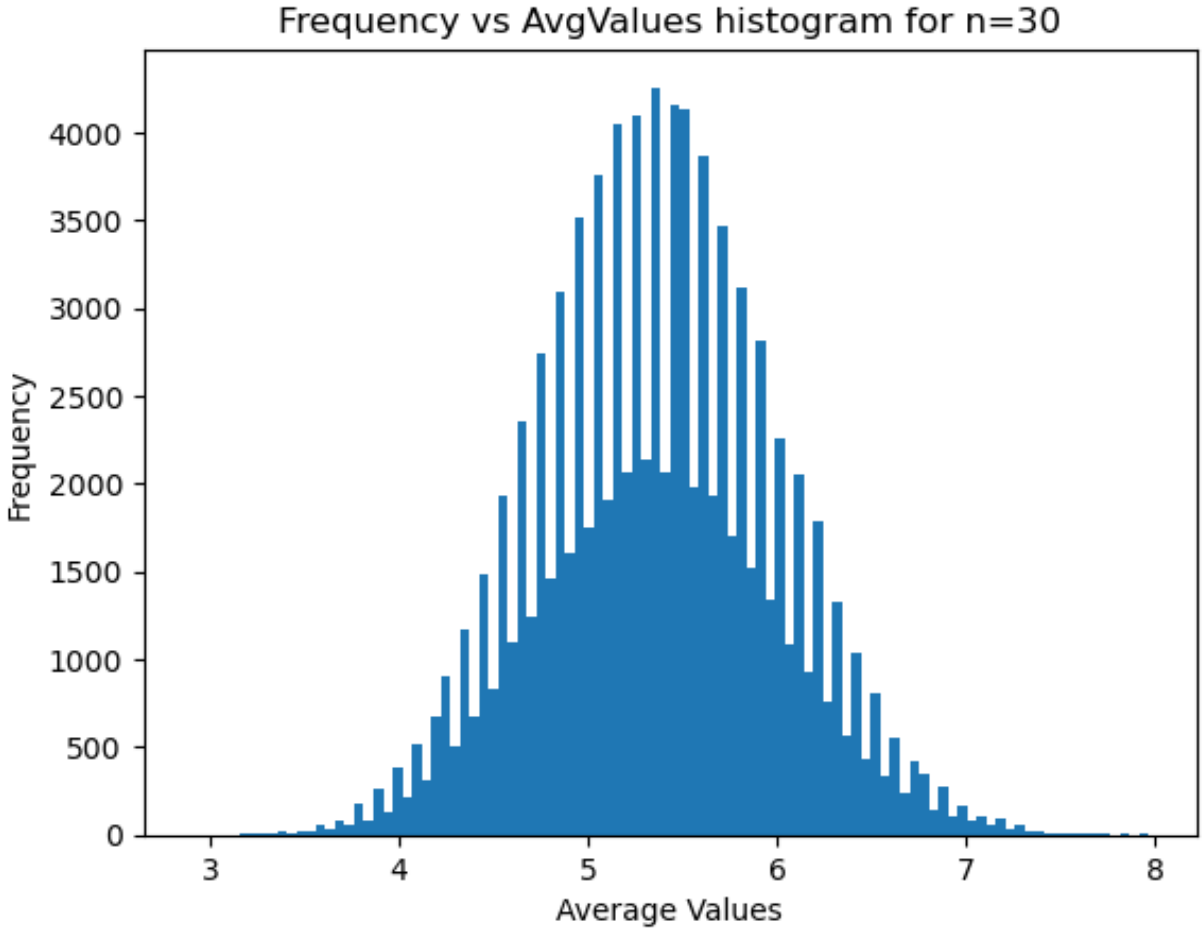
From a set $S = \{1, 1, 2, 3, 5, 5, 5, 7, 8, 10, 12\}$ of numbers, pick $n = 1$ numbers at random and compute the average. Repeat this $10^5$ times and plot the frequency of the average values (i.e., how many times each average value occurs) as a histogram. Repeat this experiment with $n \in \{5, 10, 30, 100, 1000\}$.

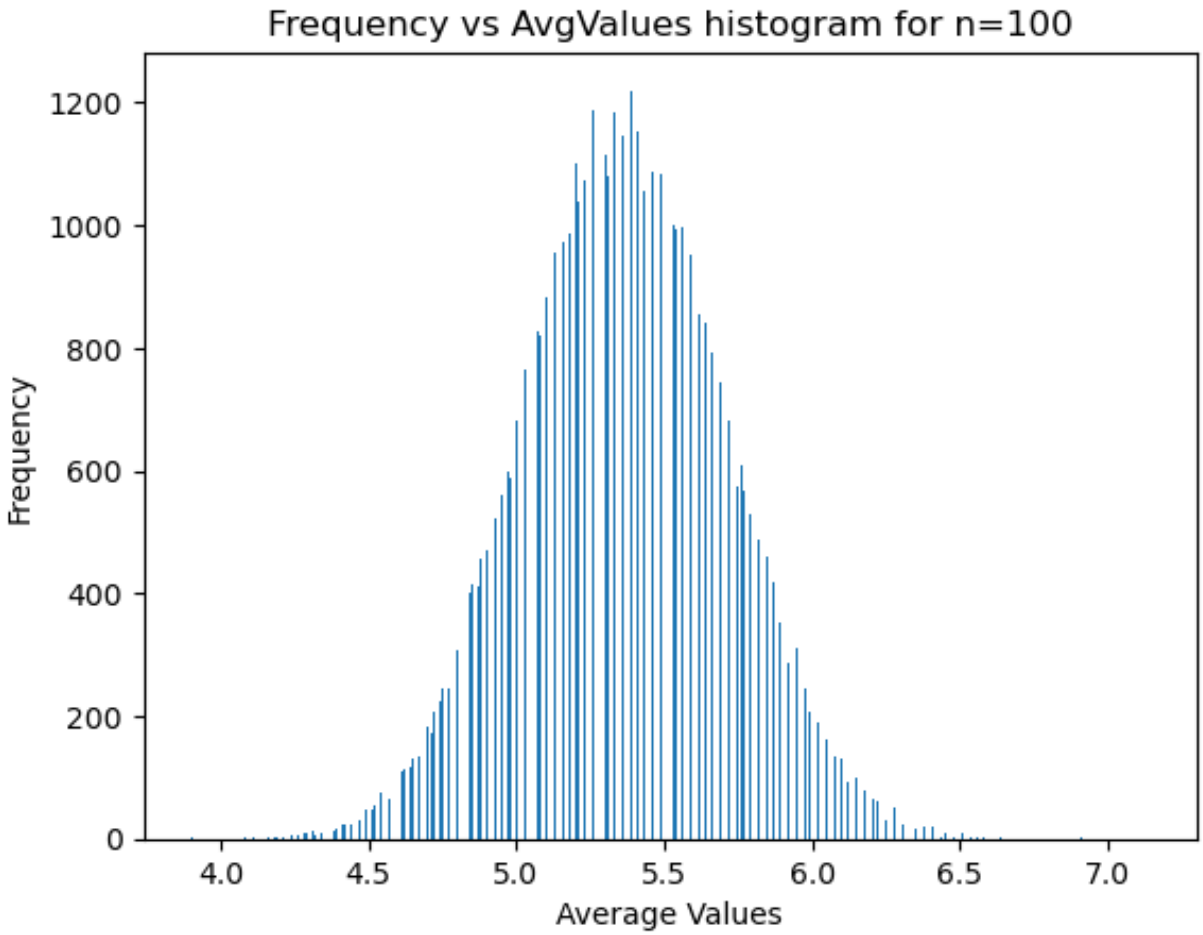
In [ ]:
```
myList = [1,1,2,3,5,5,5,7,8,10,12]
def generateIndexList(n):
    indexList = [random.randint(0,len(myList)-1) for i in range(n)]
#     to generate list of random indices
    return indexList

def pickAvg(n):
    avgValues = [0 for i in range(100000)]
    j = 0
    for i in range(100000):
        indexList = generateIndexList(n)
        sumList = 0
#         initialising sum to zero

        for index in indexList:
            sumList += myList[index]
#             finding the sum of the randomly chosen elements from the se
        avg = sumList / len(indexList)
        avgValues[j] = avg
        j+=1

#     avgValuesSet = set(avgValues)
#     freq = [0 for i in range(len(avgValuesSet))]
#     avgValuesSort = sorted(avgValuesSet)
#     j = 0
#     for value in avgValuesSort:
#         freq[j] = avgValues.count(value)
#         j+=1
    plt.hist(avgValues,bins = 100 if n<50 else 1000)
#     plotting the histogram




#     plt.hist(freq,bins = avgValuesSort)
#     fig = plt.figure()
#     fig.set_figwidth(4)
#     plt.figure(figsize=(5,5))
    plt.xlabel('Average Values')
    plt.ylabel('Frequency')
    plt.title(f'Frequency vs AvgValues histogram for n={n}')
#     plt.figure(figsize=(10,5))
#     fig = plt.figure()
#     fig.set_figwidth(7)
    plt.show()

pickAvg(1)
for n in [5,10,30,100,1000]:
    pickAvg(n)
```

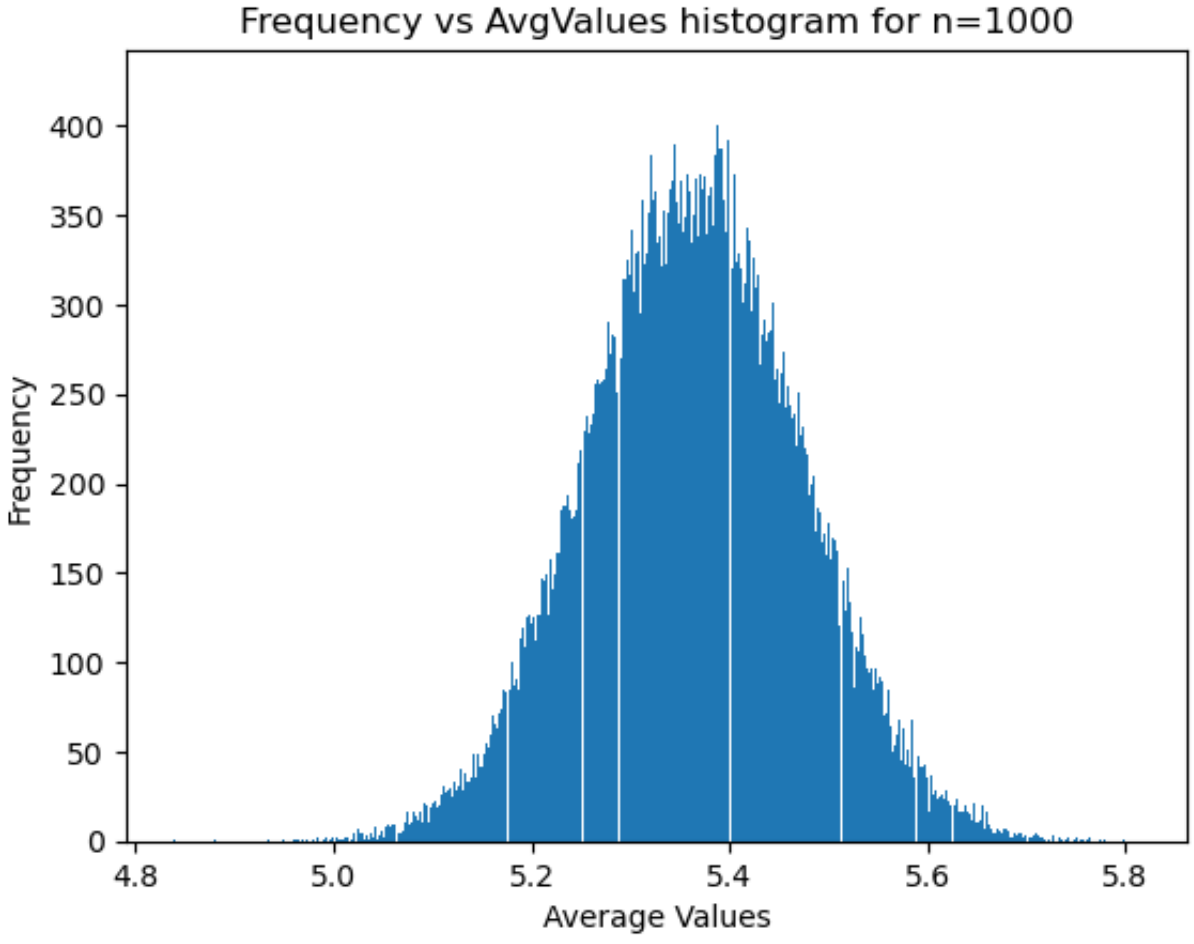

Frequency vs AvgValues histogram for n=1
Frequency
Average Values
0
5000
10000
15000
20000
25000
2
4
6
8
10
12


Frequency vs AvgValues histogram for n=5
Frequency
Average Values
0
1000
2000
3000
4000
5000
2
4
6
8
10
12


Frequency vs AvgValues histogram for n=10
Frequency
Average Values
0
500
1000
1500
2000
2500
3000
3500
2
4
6
8
10


Frequency vs AvgValues histogram for n=30
Frequency
Average Values
0
500
1000
1500
2000
2500
3000
3500
4000
3
4
5
6
7
8


Frequency vs AvgValues histogram for n=100
Frequency
Average Values
0
200
400
600
800
1000
1200
4.0
4.5
5.0
5.5
6.0
6.5
7.0


Frequency vs AvgValues histogram for n=1000
Frequency
Average Values
0
50
100
150
200
250
300
350
400
4.8
5.0
5.2
5.4
5.6
5.8

In [ ]: