Probability & Statistics
Project

Hana Ali Rashid, hr05940
Tasmiya Malik, Student ID
Ifrah Ilyas, Student ID

April 22, 2021

Q1: Random Walk

1.1

Function implementation in Python:

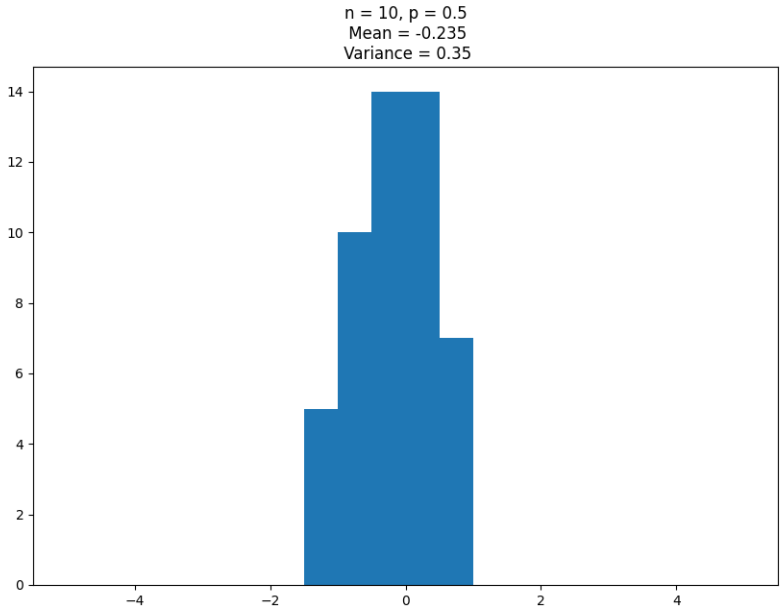
```
def get_updated_position(n,p):
    pos = 0 #position
    for _ in range(n):
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p*100:
            pos += 1 #move one step right
        else:
            pos -= 1 #move one step left
    return pos #return final position
```

Calling the function for several iterations to get multiple expected values:

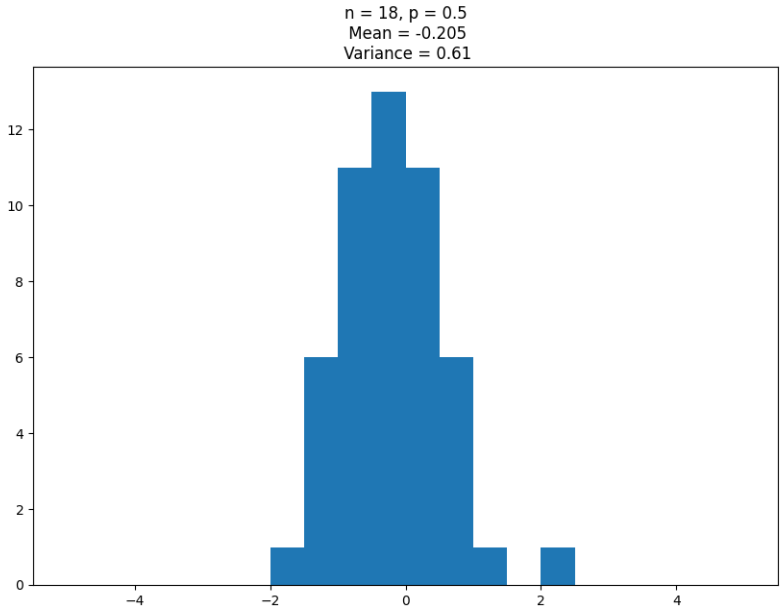
```
def main_11():
    '''Calculating expected outcomes for various combinations of n and p'''
    p = 0.7
    while p <= 0.9: #for values of p from 0.5 to 0.9
        for n in range(10,20): #for values of n from 10 to 20
            expected = []
            for j in range(50): #expected values for each (n,p) for 50 iterations
                outcomes = []
                for i in range(25):
                    outcomes.append(get_updated_position(n,p))
                expected.append(sum(outcomes)/25) #appending the expected (average) value for
    each(n,p)
            #plotting and showing a histogram of calculated expected values
            fig, ax = plt.subplots(figsize =(10, 7))
            ax.hist(expected, bins =
    [-5,-4.5,-4,-3.5,-3,-2.5,-2,-1.5,-1,-0.5,0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,1

            plt.title('n = '+str(n)+', p = '+str(p)+'\n Mean = '+str(round(statistics.mean(
    expected),3))+'\n Variance = '+str(round(statistics.variance(expected),3)))
            plt.savefig("Q1_histograms/q1"+'_n = '+str(n)+'_ p = '+str(p)+'.png')
            plt.show()
        p = round((p+0.1),1) #incrementing
```

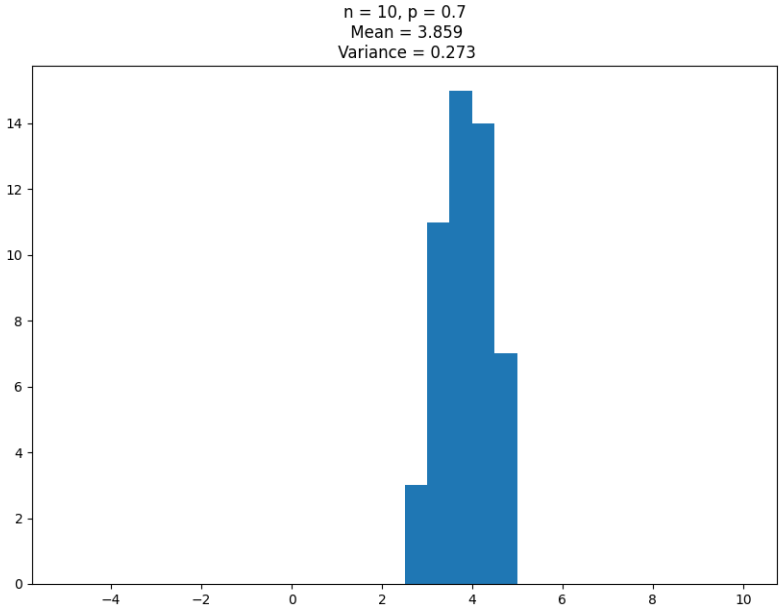
Histograms produced by the above code for various combinations of n and p:



The above histogram appears to follow a normal distribution with a mean of 3.267 and variance of 0.233.



The above histogram appears to follow a normal distribution with a mean of 3.267 and variance of 0.233.



The above histogram appears to follow a normal distribution with a mean of 3.267 and variance of 0.233.

1.2

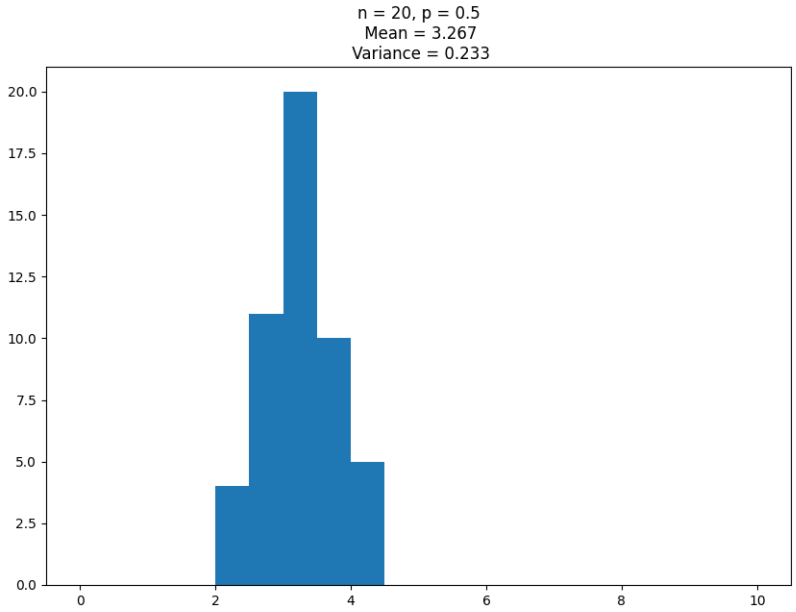
Function implementation in Python:

```
def get_updated_position_restricted(n,p):
    pos = 0 #position
    for _ in range(n):
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p*100 or pos <= 0: #move one step right if pos == 0
            pos += 1
        else:
            pos -= 1 #move one step left
    return pos #return final position
```

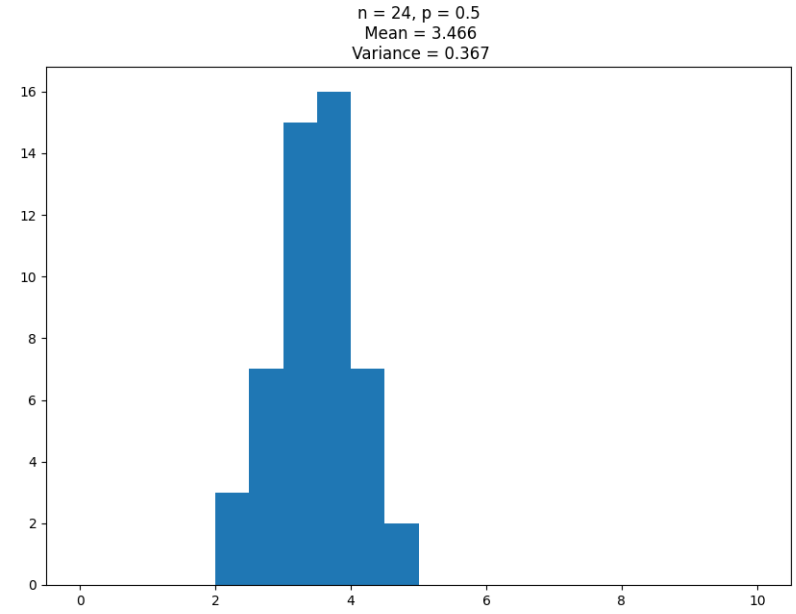
Calling the function for several iterations to get multiple expected values:

```
def main_12():
    '''Calculating expected outcomes for various combinations of n and p'''
    p = 0.7
    while p <= 0.9:
        for n in range(20,30):
            expected = []
            for j in range(50): #expected value for each (n,p) for 25 iterations
                outcomes = []
                for i in range(25):
                    outcomes.append(get_updated_position_restricted(n,p))
                expected.append(sum(outcomes)/25) #appending the expected (average) value for
    each(n,p)
            #plotting and showing a histogram of calculated expected values
            fig, ax = plt.subplots(figsize =(10, 7))
            ax.hist(expected, bins =
    [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10])
            plt.title('n = '+str(n)+', p = '+str(p)+'\n Mean = '+str(round(statistics.mean(
    expected),3))+'\n Variance = '+str(round(statistics.variance(expected),3)))
            plt.savefig("Q1_histograms/Q1.2 "+'_n = '+str(n)+'_p = '+str(p)+'.png')
            plt.show()
        p = round((p+0.1),1) #incrementing
```

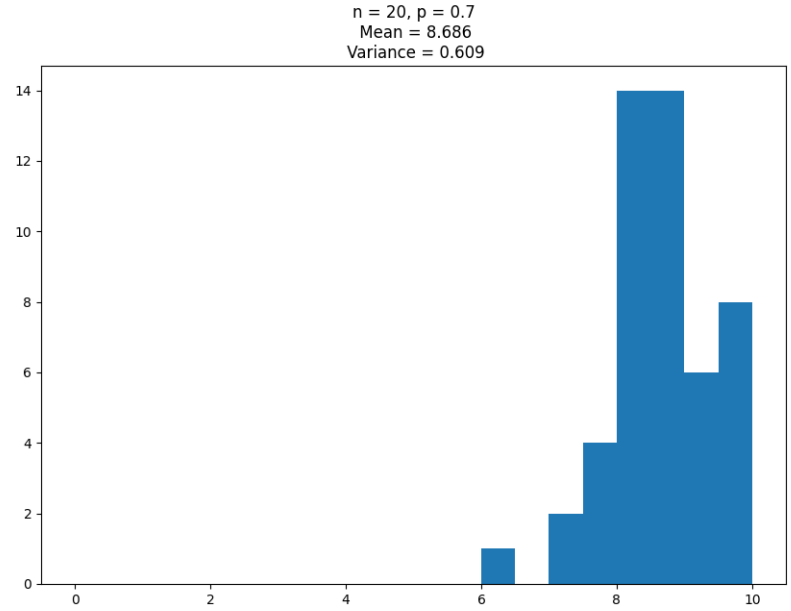
Histograms produced by the above code for various combinations of n and p:



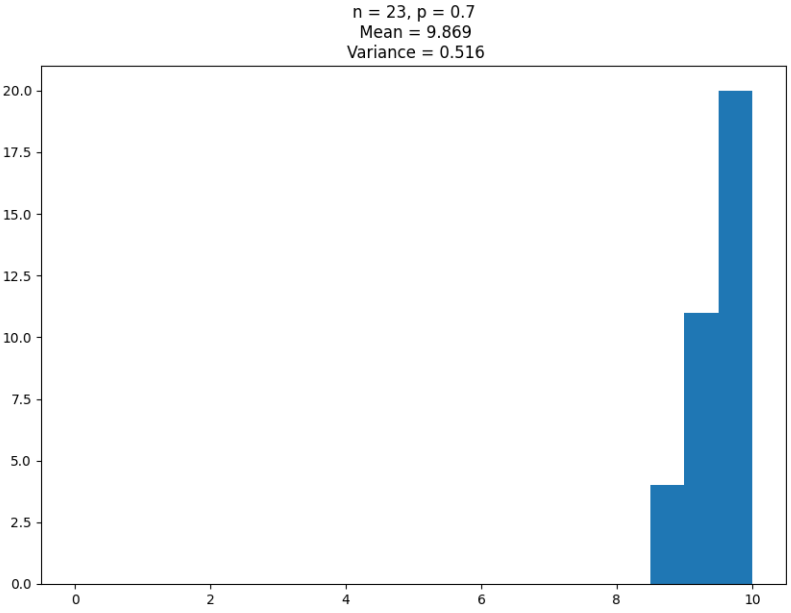
The above histogram appears to follow a normal distribution with a mean of 3.267 and variance of 0.233.



The above histogram shows that...



The above histogram shows that...



1.3

Function implementation in Python:

```
def stepsToMeet(pos1,pos2,p1,p2):
    count = 0 #keeps count of number of steps taken for objects to meet
    while pos1 != pos2:
        rand = random.randint(1,100) #generating a random number in the range 1 to 100 to
    determine outcome
        if rand < p1*100:
            pos1 += 1 #move one step right
        else:
            pos1 -= 1 #move one step left
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p2*100:
            pos2 += 1 #move one step right
        else:
            pos2 -= 1 #move one step left
        count += 1
    return count
```

Calling the function for several iterations to get multiple expected values:

```
def main_13():
    '''Calculating expected outcomes for various combinations of n & p'''
    expected = []
    outcomes = []
    p1 = 0.5
    p2 = 0.5
    pos1 = -5
    pos2 = 6
    while p1 <= 0.9:
        while p2 <= 0.9:
            for i in range(25):  #calculating the expected value for each (n,p) for 25
    iterations
                for j in range(25):
                    outcomes.append(stepsToMeet(pos1,pos2,p1,p2))
                #calculating the average expected value for each(n,p)
                expected.append(sum(outcomes)/25) #appending the expected (average) value for
    each(n,p)
                outcomes = [] #resetting outcomes list
            #plotting a histogram of calculated expected values
            fig, ax = plt.subplots(figsize =(10, 7))
            ax.hist(expected, bins = range(-10,10))
            plt.title('p1 = '+str(p1)+', p2 = '+str(p2)+', pos1 = '+str(pos1)+', pos2 = '+str(
    pos2))
            plt.show()
            expected = [] #reset expected list
            p2 = round((p2+0.1),1) #incrementing
        p1 = round((p1+0.1),1) #incrementing
```