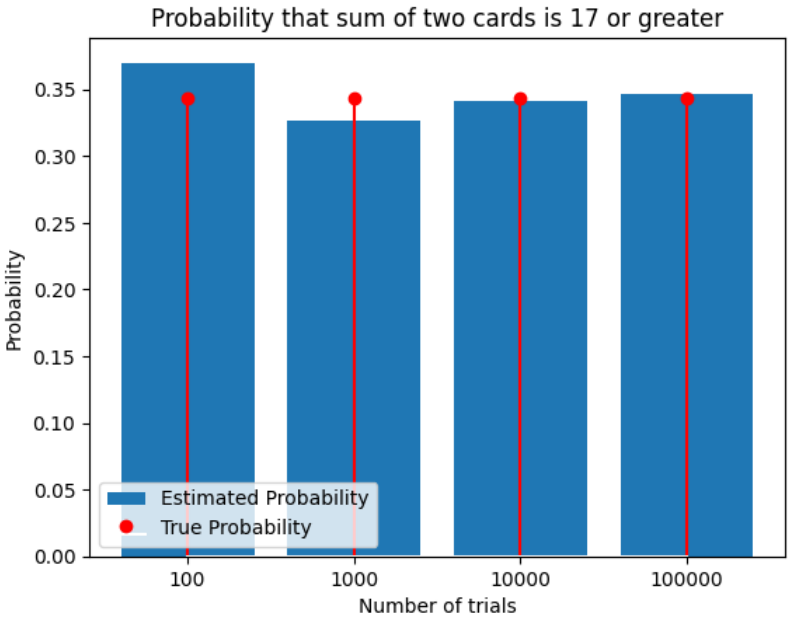
The use of simulation to generate probabilties for counting events. We know that, by counting the number of times that certain combinations of outcomes occur, we can compute the probability that those outcomes occur analytically. This is typically done when analyzing card games such as Blackjack or Poker, where one computes the probability of getting a set of cards that satisfies some criterion. An alternative is to simulate the experiment, and observe the fraction of times that the cards satisfy the criteria. We implement the approach below for three problems, and compare the simulation results with the analytical computations of probability, for different lengths of experiments.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
```

Part a. Run the program below to estimate the probability that the first five cards of a randomly shuffled deck have no more than two clubs. Remember that clubs are cards numbered 1-13. Do it for numtrials = 100, 1000, 10000, 100000. Plot the bar chart of probability estimate versus number of trials. Plot as a stem plot the true probability, which is (nchoosek(39,5) + 13*nchoosek(39,4) + nchoosek(13,2)*nchoosek(39,3))/nchoosek(52,5).

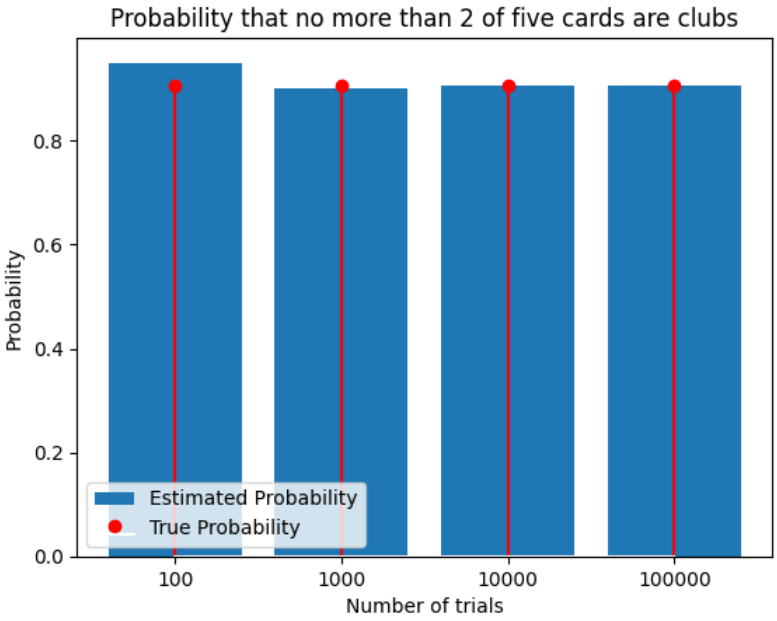
In [ ]:
```
N = 52 # number of cards
trials = [100,1000,10000,100000]  # number of cases
probability = np.zeros(4,);
cards = list(range(1,53)) # a list with elements 1, 2, ..., 52

for i in range(4):
    numtrials = trials[i]
    successes = 0

    for k in range(numtrials):

        p = np.random.permutation(cards) # generates random permutation of numbers from 1 to 52: a shuffle

        total = 0
        first = (p[0]-1) % 13     # modular arithmetic
        if (first == 0):
            total = total + 11
        elif (first >= 9):
            total = total + 10
        else:
            total = total + first + 1

        second = (p[1]-1) % 13
        if (second == 0):
            total = total + 11
        elif (second >= 9):
            total = total + 10
        else:
            total = total + second + 1

        if (total >= 17) and (total <= 21):
            successes =  successes + 1

    probability[i] = successes/numtrials

trueProb = (4*32 + 4*16 + 4*20 + 4* 16 + math.comb(16,2))/math.comb(52,2)



fig = plt.figure()
plt.bar(np.arange(4),probability)
plt.xticks(np.arange(4), trials)

plt.xlabel('Number of trials')
plt.ylabel('Probability')
plt.stem(np.arange(4),trueProb*np.ones(4,),linefmt='r',markerfmt='ro',basefmt='w')
plt.title(f"Probability that sum of two cards is 17 or greater")
plt.legend(['Estimated Probability','True Probability'])
plt.show()
fig.savefig("P3_5a.png",bbox_inches='tight')
```



Part b. Write a program to estimate the probability that the first five cards of a randomly shuffled deck have no more than two clubs. Remember that clubs are cards numbered 1-13. Do it for numtrials = 100, 1000, 10000, 100000. Plot the bar chart of probability estimate versus number of trials. Plot as a stem plot the true probability, which is (nchoosek(39,5) + 13*nchoosek(39,4) + nchoosek(13,2)*nchoosek(39,3))/nchoosek(52,5).

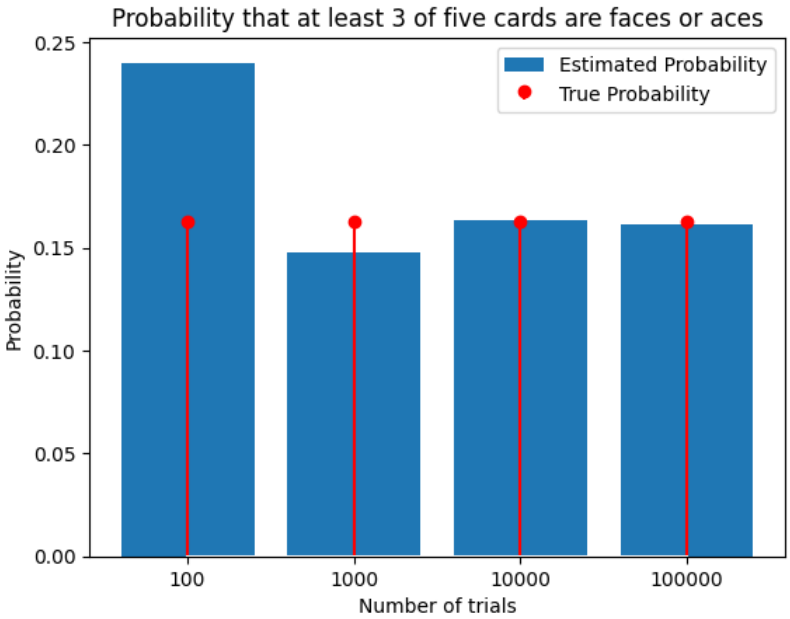
In [ ]:

```
N = 52 # number of cards
trials = [100,1000,10000,100000]  # number of cases
probability = np.zeros(4,);
cards = list(range(1,53)) # a list with elements 1, 2, ..., 52

for i in range(4):
    numtrials = trials[i]
    successes = 0

    for k in range(numtrials):
        p = np.random.permutation(cards) # generates random permutation of numbers from 1 to 52: a shuffle

        #%%% Write your code here: You have to add a success if
        #%%% the number of clubs in the first five cards is less than or
        #%%% equal to 2.  See how it was done in the previous part.

        count_clubs = 0
        for l in range(5):
            if p[l] <= 13:
                count_clubs += 1

        if count_clubs <= 2:
            successes += 1

    probability[i] = successes/numtrials

trueProb = (math.comb(39,5)+ 13*math.comb(39,4) + math.comb(13,2)*math.comb(39,3))/math.comb(52,5)


fig = plt.figure()
plt.bar(np.arange(4),probability)
plt.xticks(np.arange(4), trials)

plt.xlabel('Number of trials')
plt.ylabel('Probability')
plt.stem(np.arange(4),trueProb*np.ones(4,),linefmt='r',markerfmt='ro',basefmt='w')
plt.title("Probability that no more than 2 of five cards are clubs")
plt.legend(['Estimated Probability','True Probability'])
plt.show()
fig.savefig("P3_5b.png",bbox_inches='tight')
```



Part c. Write a program to estimate the probability that you get 3 or more face cards or aces (jacks, queens, kings, aces) in the first five cards of a randomly shuffled deck. Remember that, if a card is numbered n, then (n-1) mod 13 = 0 for aces and (n-1) mod 13 >= 10 for jacks, queens and kings. Do it for numtrials = 100,1000, 10000, 100000. Plot the bar chart of probability estimate versus number of trials. Plot as a stem plot the true probability, which is (nchoosek(16,5)+ nchoosek(16,4)*nchoosek(36,1) + nchoosek(16,3)*nchoosek(36,2))/nchoosek(52,5).

In [ ]:
```
N = 52 # number of cards
trials = [100,1000,10000,100000]  # number of cases
probability = np.zeros(4,);
cards = list(range(1,53)) # a list with elements 1, 2, ..., 52

for i in range(4):
    numtrials = trials[i]
    successes = 0

    for k  in range(numtrials):

        p = np.random.permutation(cards) # generates random permutation of numbers from 1 to 52: a shuffle

        # %%% Write your code here: You have to add a success if
        # %%% the number of face cards or aces in the first in the first
        # %%% five cards is at least 3.
        # %%% See how it was done in the part (a).

        count_faces_or_ace = 0
        for l in range(5):
            if (p[l] - 1) % 13 == 0 or (p[l] - 1) % 13 >= 10:
                count_faces_or_ace += 1

        if count_faces_or_ace >= 3:
            successes += 1

    probability[i] = successes/numtrials

trueProb = (math.comb(16,5) + math.comb(16,4)*36 + math.comb(16,3)*math.comb(36,2))/math.comb(52,5)


fig = plt.figure()
plt.bar(np.arange(4),probability)
plt.xticks(np.arange(4), trials)

plt.xlabel('Number of trials')
plt.ylabel('Probability')
plt.stem(np.arange(4),trueProb*np.ones(4,),linefmt='r',markerfmt='ro',basefmt='w')
plt.title(f"Probability that at least 3 of five cards are faces or aces")
plt.legend(['Estimated Probability','True Probability'])
plt.show()
fig.savefig("P3_5c.png",bbox_inches='tight')
```



Part d. Part c is more difficult to predict than parts a or b because the event in part c is intrinsically more complex and more variable than the events in parts a and b. There are more, complex combinations of cards that satisfy the event in part c.