Problem Zero: The Python Random Library

Go to https://docs.python.org/2/library/random.html and read about the random library. We will use the following two functions, which you should understand well before proceeding:

```python
In [5]:
1 import random
2 # random.randint(a,b)
3 random.randint(0, 10)
4 # random.random()
5 random.random()
```

Out[5]: 0.6963483384865372

Problem One: Verification of Randomness

In this problem, we are going to verify that the randint(...) function produces a sequence of integers in the range [a..b] which appear to be truly random, in that they can pass various "tests of randomness." For each, we can imagine that we are calling the function a very large number of times (e.g., \(10^6\) times), and examining the sequence of \(10^6\) random numbers for signs of randomness.

(a) The first test of randomness (the "spectral test") is that if we generate a large number of random integer values in the range [a..b], each number in this range should occur about the same number of times. For example, if we generate \(N = 10^6\) numbers in the range [0..9], we should see about \(10^5\) of each number (this is called a "uniform distribution"). Write a function randomness_test1(N) which tests this assumption, i.e., declare a list \(X = [0]*10\) which will record how many times each of the digits 0 .. 9 is generated in \(10^6\) calls to randint(...), and prints out the array \(X\) at the end.

Answer this question in a comment at the end of your code: Does the function randint(...) pass this test?

```python
In [6]:
1 # Yes, randint() pass the test since the number of occurrence of each digit is similar

The number of times each digit appears:
[100416, 99454, 99741, 100259, 99339, 100107, 100597, 99969, 100212, 99906]
```

(b) Another test of randomness is called the Permutation Test -- If we take pairs of successive numbers (e.g., if the sequence is .2, .3,.5,.14, .7,.01, ..., then you would consider (.2,.3), (.5,.14), (.7,.01), etc.) and compare the relative ordering of a pair \((x,y)\) then there should be about 50% with \(x < y\) and 50% with \(y < x\) (the chance that \(x = y\) is very, very small, and can be ignored); similarly, if we take successive sequences of \(K\) numbers, the relative ordering of the \(K\)-tuples should be uniformly distributed. Let us just consider the case of pairs of numbers and again use random.random() to generate a sequence of floating point numbers in the range [0 .. 1). Write a function randomness_test2(N) which tests this assumption, i.e., declare a list \(X = [0]*2\) how many times each of the digits 0 .. 9 is generated in \(10^6\) calls to randint(...), and prints out the array \(X\) at the end.

Answer in a comment at the end of your code: Does the function random() pass this test?

```python
In [7]:
1 def randomness_test2(N):
2     pass
3
4 print("The number of ascending pairs and descending pairs: ")
5 randomness_test2(10**6)
6
7 # Yes, random() pass the test since the number of ascending pairs and the number of descending pairs are similar
8 # pairs are similar

The number of ascending pairs and descending pairs:
[499769, 500231]
```

Problem Two: Simulation of Dice Rolls

In this problem we will develop various functions which simulate random experiments. These should, of course, correspond to our theoretical understanding of the same experiments using probability theory.
(a) Write a function `dieRoll(N)` which simulates rolling a single die \(N\) times, and which prints out the probability, based on the experiment, that each of the numbers 1, 2, ..., 6, appeared. To calculate the probabilities, take the number of times each number appeared, and divide by \(N\). (This is the “frequentist” interpretation of probability, that probabilities are determined from such practical experience with random processes.) Run the function for \(N = 10, 10^3, \) and \(10^6\).

Answer at the end of your solution in a comment: What do you observe as \(N\) increases?

In [8]:

```python
def dieRoll(N):
    pass

print("Probability of each outcome, 10 experiments: ")
dieRoll(10)
print("Probability of each outcome, 10^3 experiments: ")
dieRoll(10**3)
print("Probability of each outcome, 10^6 experiments: ")
dieRoll(10**6)

# As N increases, the possibility of each outcome gets closer to 1/6
```

Probability of each outcome, 10 experiments:
[0.1, 0.0, 0.3, 0.1, 0.0, 0.5]
Probability of each outcome, 10^3 experiments:
[0.156, 0.173, 0.168, 0.169, 0.167, 0.167]
Probability of each outcome, 10^6 experiments:
[0.166614, 0.166211, 0.166704, 0.167022, 0.166764, 0.166685]

(b) Now write a function `twoDiceRolls(N)` which prints out the probability of each of the outcomes 2, 3, ..., 12 for the sum of the dots in a roll of two independent dice (or two independent rolls of a single die), over \(N\) trials. Use \(N = 10^6\) iterations of a double roll to do the calculations.

Answer at the end of your solution in a comment: Is this equiprobable? Explain why or why not.

In [9]:

```python
def twoDiceRolls(N):
    pass

print("Probability of each outcome: ")
twoDiceRolls(10**6)

# This is not equiprobably. There are fewer ways to get some numbers (for instance, 2, 
# which can only appear in one way) than others (like 7, which can appear in multiple ways)
```

Probability of each outcome:
[0.027708, 0.055405, 0.083404, 0.111111, 0.139209, 0.167283, 0.138359, 0.111232, 0.08295, 0.055623, 0.027716]

**Problem Three: Simulation of Coin Flips (fair and unfair)**

Now suppose that you have a coin, but it is NOT necessarily a fair coin: you only know that the chance of getting heads is \(p\) and the chance of tails is \(1-p\). You can simulate this by writing a function `nextFlip(p)` which returns true (heads) or false (tails) and proceeds by using `random.random()` and checking if the returned value is < \(p\) (heads) or \(>= p\) (tails). (Note: the chance of getting exactly \(p\) is so small we could use < or \(<=\) with little difference in the results).

For each of these problems, you will run the same experiment for \(p = 0.5\) (i.e., a fair coin), and \(p = 0.7\) and \(N\) (the number of times you try the experiment) will be \(10^6\).

(a) Set \(p = 0.5\) and flip the coin \(N = 10^6\) times, and then repeat the experiment for \(p = 0.7\). Code your solution as a function `coinSimulation(p,N)`.

Answer in a comment: Explain what you see by expressing the number of heads and the number of tails as a function of \(N\) and \(p\).
In [10]:
def nextFlip(p):
    pass

def coinSimulation(p, N):
    pass

# when p = 0.5
8 print("The number of heads and the number of tails with p = 0.5")
9 coinSimulation(0.5, 10**6)

# when p = 0.7
11 print("The number of heads and the number of tails with p = 0.7")
12 coinSimulation(0.7, 10**6)

# the number of heads = p * N
14 # the number of tails = (1-p) * N

The number of heads and the number of tails with p = 0.5
500409, 499591
The number of heads and the number of tails with p = 0.7
699683, 300317

(b) Set p = 0.5. Suppose we call it a "trial" when you flip this coin 4 times and count the number of heads that appears in
these 4 flips. Code your solution to this as a function coinTrial(p). Now our question is: what is the probability of K heads in a trial?
To answer this, perform N = 10^6 trials (each of which consists of flipping the coin 4 times and recording how many heads
appear). Keep track of the results in an array X = [0]*5 (you need 5 totals for each of K = 0, 1, 2, 3, 4). Then divide each of these 5
totals by N = 10^6 to obtain the probability of each K. Repeat the experiment for p = 0.7. Code your solution as a function
coinExperiment(p,N).

Answer in a comment: Explain why the results are different for p = 0.5 and p = 0.7.

In [11]:
def coinTrial(p):
    pass
def coinExperiment(p, N):
    pass

# when p = 0.5
8 print("Probability of K heads when p = 0.5")
9 coinExperiment(0.5, 10**6)

# when p = 0.7
11 print("Probability of K heads when p = 0.7")
12 coinExperiment(0.7, 10**6)

# The higher the probability of getting a head is, the more heads we expect to see
# when we flip the coin 4 times.
14 # Therefore, higher numbers of heads are more likely when p=.7 than when
17 # p=.5 and lower numbers of heads are less likely.

Probability of K heads when p = 0.5
[0.062569, 0.250704, 0.374691, 0.249565, 0.062471]
Probability of K heads when p = 0.7
[0.00802, 0.07556, 0.264469, 0.411227, 0.240724]

(c) Now we will perform the experiment mentioned in class several times: How many flips does it take to get the first head? The
sample space is infinite, i.e., S = { 1, 2, 3, 4, .... }, but we will only keep track of results in the range [1 .. 20], ignoring any cases
where it takes more than 20 flips (very unlikely in any case). Write a short function which performs this experiment 10^6 times and
prints out the probabilities for outcomes in the range [1 .. 20] in two cases: (i) when p = 0.5 and (ii) when p = 0.7. Your results
should confirm the theoretical result that P(k) = (1-p)^(k-1)*p (i.e., in the case p = 0.5, you should get 0.5, 0.25, 0.125, etc., and for
p = 0.7 you should get 0.7, 0.21, 0.063, .... ). Code your solution as a function firstHead(p,N).

Answer at the end in a comment: Explain why the results are different for p = 0.5 and p = 0.7, i.e., why does the sequence
converge to 0 more quickly for 0.7?
Problem Four: Let's Gamble!

Now suppose we want to actually figure out the right way to gamble, using a computer simulation. We'll go back to dice....

We will play a version of blackjack and figure out the best strategy for the game "Twenty One."

This game is similar to the famous card game blackjack. We will play a one-player version of the game. The game is played for some number \( N \) of rounds (we will use \( N = 10^6 \)), at the end of which the player wins points. The player accumulates points during the whole game, and the objective is, of course, to end up with the maximum number of points.

The objective in each round of the game is to score as close to 21 as possible by rolling a die as many times as you wish and adding all the numbers that appear. When a player's total exceeds 21, he is 'busted' and gets 0 points. If the player chooses to stop rolling before he exceeds 21, then the sum of the numbers rolled in that round is the number of points he wins. (There are many variations on this game, some involving multiple players, or a "banker" or different numbers of dice, or alcohol..... here is a short YT video explaining the basic game.)

A computer can play this game with a suitable strategy. For this problem, we will consider a strategy to be simply an integer \( K \) which is the value at which you stop rolling (thinking that you are close enough to 21). The number \( K \) is fixed for the entire game. For example, if you set \( K = 19 \), then in every round, you will keep rolling if your sum to that point is less than or equal to 19; if you get more than 19 you stop.

You should write a function \( \text{playRound}(K) \) which rolls a single die until you reach \( K \) or get busted, and either return your score (if you reached \( K \)), or 0 (if you were busted). Then write a function \( \text{playGame()} \) which calls \( \text{playRound}(K) \) for \( N = 10^6 \) times for each \( K \) and returns an array of 21 numbers giving the average payoff for each \( K = 1, \ldots, 21 \).

Your task is to answer the following questions in a comment at the end of your code:

- For each \( K = 1 .. 21 \), what is the average payoff per round for a game played with this strategy?
- What is the best strategy for the game, meaning what value of \( K \) wins the most points on average?

Hint: For the first question, write \( \text{playGame()} \) with a double loop that tries each value of \( K \) for \( N \) rounds:

```python
for K in range(1,22):
    for i in range(N):
        #call playRound(K) and record the results
        #divide your totals in the array by N to get the average payoff for each K
```
In [14]:
   1  def playRound(K):
   2     pass
   3
   4  def playGame(N):
   5     pass
   6
   7  playGame(10**6)


# Best score is obtained for K = 16