Discussion Section Exercises for October 22-23

These are all loosely inspired by the dice example presented in the class, although the details are different. The last problem is trickier. All but the last can be handled with a single line of code, if you use list comprehension, but it is quite acceptable to do this other ways, building the list incrementally using either the append method or concatenation.

List methods that could be useful here:

```plaintext
append
reverse
sort
count
```

as well as the sum function. You call the sum function as `sum(L)` but you call the other methods as `L.methodname()`. The append, reverse and sort methods change the list.

1. Write a function `dicerolls(k)` that returns a list of the results of k rolls of a single die. Just to get you moving in the right direction (and solving most of the problem for you) if I want to create a list of 10,000 rolls of a single die, I could do this either with a for statement, as

```plaintext
mylist=[]
for j in range(10000):
    mylist.append(random.randint(1,6))
```

Or, alternatively, using list comprehension:

```plaintext
mylist = [random.randint(1,6) for j in range(10000)]
```

So all you have to do is package this in a function, substituting the parameter `k` for the value 10000. When you are finished, you should be able to type

```plaintext
v=dicerolls(20)
```

and then print(v) to see a list of the 20 dice rolls. In one trial I got

```plaintext
[2, 2, 4, 6, 4, 6, 4, 1, 6, 2, 2, 2, 6, 5, 3, 6, 3, 2, 5, 3]
```

but of course you will get a different answer every time you run it.

2. Now write a function `dicetriples(k)` that returns the results of k rolls of three dice. Each result is itself represented by a list of the three component outcomes, so this is a list of lists. For example, calling
v=dicetriples(20)

and then printing v gives an output like:

\[
[[2, 2, 3], [1, 3, 3], [4, 3, 6], [6, 5, 1], [1, 3, 4],
[4, 6, 1], [6, 3, 6], [6, 5, 5], [3, 6, 1], [2, 4, 3],
[5, 3, 6], [1, 2, 2], [4, 4, 1], [3, 5, 2], [1, 2, 1],
[5, 3, 1], [4, 6, 6], [2, 3, 6], [6, 5, 5], [1, 2, 4]]
\]

There are many ways to go about it. It can be done in a single line, if you are clever about nesting list comprehensions, or with a for loop, or nested for loops...

3. Now write a function three_dice(k) that calls the function from Problem 2, and produces a list of the sums of k rolls of three successive dice. (It’s an important part of this problem that you do this by first calling the result of Problem 2, rather than starting from scratch.) A typical run with k=20 produced this output.

\[7, 11, 15, 11, 8, 10, 12, 10, 9, 7, 13, 11, 12, 16, 9, 11, 9, 11, 8\]

Use this with some large value of k (like one hundred thousand) to estimate the probability that the sum of the values on three dice is 7. As the experiment described below suggests, the answer will be around 0.068. (The theoretically predicted value is 5/72=0.06944.)

4. Now write a function three_dice_sums(k) that calls the function in Problem 3 and returns a list of length 16 that gives the number of occurrences of each of the 16 possible outcomes, from 3 through 18. Each element in the returned list should have the form [j,n] where n is the number of times j occurred as the sum of the three dice. For example, when I ran this with k =10000, I got

\[[[3, 3], [4, 127], [5, 282], [6, 463], [7, 676], [8, 980],
[9, 1156], [10, 1258], [11, 1230], [12, 1186], [13, 974],
[14, 690], [15, 464], [16, 262], [17, 161], [18, 58]]\]

You’ll note that this is very much like what we did in class with rolls of two dice, except we obtained the result a bit differently.
5. Finally, write a function \texttt{threedicesums_sorted(k)} that calls the result of Problem 4, but returns the list of outcomes in order of increasing frequency: In other words, the counts for the least frequent occurrence (which will probably be 3 or 18) come first. By the way, if you sort the list from Problem 4 just using the \texttt{L.sort()} method, you will just get the same list back, because the elements of L are compared using the first component. So it is a bit tricky to get around this! Here is what results of a sample run with \(k=10000\) look like. (You should get different, but similar, values.)

\[
[[3, 50], [18, 66], [17, 128], [4, 141], [16, 250], [5, 273], [6, 460], [15, 469], [7, 696], [14, 701], [8, 987], [13, 997], [9, 1097], [12, 1184], [10, 1243], [11, 1258]]
\]

Helpful Hint: There is a list method \texttt{reverse()} that reverses a list: So if \(s\) is a list, \(s.reverse()\) changes the list \(s\) to the reversed version, e.g.,

\[
\begin{align*}
\text{>>s=}[1,2,3] \\
\text{>>s.reverse()} \\
\text{>>print(s)} \\
[3,2,1]
\end{align*}
\]