IMPORTANT: When submitting this homework notebook, please modify only the cells that start with:

```
# modify this cell
```

Different Dice

So far we mostly considered standard 6-faced dice. The following problems explore dice with different number of faces.

In [1]:
```
import numpy as np
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Problem 1

Suppose that a 6-sided die is rolled n times. Let X_i be the value of the top face at the ith roll, and let $X \triangleq \max_{1 \le i \le n} X_i$ be the highest value observed. For example, if $n = 3$ and the three rolls are 4, 1, and 4, then $X_1 = 4, X_2 = 1, X_3 = 4$ and $X = 4$.

To find the distribution of X, observe first that $X \le x$ iff $X_i \le x$ for all $1 \le i \le n$, hence $P(X \le x) = (x/6)^n$. It follows that $P(X = x) = P(X \le x) - P(X \le x - 1) = (x/6)^n - ((x-1)/6)^n$. For example, $P(X = 1) = (1/6)^n$, and $P(X = 2) = (1/3)^n - (1/6)^n$.

In this problem we assume that each of the n dice has a potentially different number of faces, denoted f_i, and ask you to write a function **largest_face** that determines the probability $P(x)$ that the highest top face observed is x. **largest_face** takes a vector $\boldsymbol{f}$ of positive integers, interpreted as the number of faces of each of the dice, and a value x and returns $P(x)$. For example, if $\boldsymbol{f} = [2, 5, 7]$, then three dice are rolled, and $P(1) = (1/2) \cdot (1/5) \cdot (1/7)$ as all dice must be 1, while $P(7) = 1/7$ as the third die must turn up 7.

Sample run

```
print largest_face([2,5,8],8)
print largest_face([2], 1)
largest_face( [3,4], 2)
print largest_face([2, 5, 7, 3], 3)
```

Expected Output

```
0.125
0.5
0.25
0.180952380952
```

In [2]:
```
# modify this cell

def largest_face(f, x_max):
    # inputs: m is a list of integers and m_max is an integer
    # output: a variable of type 'float'

    #
    # YOUR CODE HERE
    #
```

In [3]:
```
# Check Function

assert abs( largest_face([5],3) - 0.19999999999999996 ) < 10**-5
assert abs( largest_face( [11,5,4], 5) - 0.16363636363636364  ) < 10**-5
assert abs( largest_face(range(1,10), 3) - 0.011348104056437391 ) < 10**-


#
# AUTOGRADER TEST - DO NOT REMOVE
#
```

Problem 2

Write a function **face_sum** that takes a vector f that as in the previous problem represents the number of faces of each die, and a positive integer s, and returns the probability that the sum of the top faces observed is s. For example, if $f = [3, 4, 5]$ and $s \leq 2$ or $s \geq 13$, **face_sum** returns 0, and if $s = 3$ or $s = 12$, it returns $(1/3) \cdot (1/4) \cdot (1/5) = 1/60$.

Hint: The **constrained-composition** function you wrote for an earlier probelm may prove handy.

Sample run

```
print face_sum([3, 4, 5], 13)
print face_sum([2,2],3)
print face_sum([3, 4, 5], 7)
```

Expected Output

```
0.0
0.5
0.18333333
```

Helper Code

Below is a correct implementation of **constrained_composition**. Call this function in your implementation of **face_sum**.

In [4]:
```
def constrained_compositions(n, m):
    # inputs: n is of type 'int' and m is a list of integers
    # output: a set of tuples

    k = len(m)
    parts = set()
    if k == n:
        if 1 <= min(m):
            parts.add((1,)*n)
    if k == 1:
        if n <= m[0]:
            parts.add((n,))
    else:
        for x in range(1, min(n-k+2,m[0]+1)):
            for y in constrained_compositions(n-x, m[1:]):
                parts.add((x,)+y)
    return parts
```

exercise:

In [5]:
```
# modify this cell

def face_sum(m, s):
    # inputs: m is list of integers and s is an integer
    # output: a variable of type 'float'

    #
    # YOUR CODE HERE
    #
```

In [6]:
```
# Check Function
assert sum(abs( face_sum([2,2],2) - 0.25  )) < 10**-5
assert sum(abs( face_sum([2,2],10) - 0.0  )) < 10**-5
assert sum(abs( face_sum(range(1,10),20) - 0.03037092151675485  )) < 10**

#
# AUTOGRADER TEST - DO NOT REMOVE
#
```

In [ ]:

```

```