STATISTICAL RETHINKING 2024
WEEK 1 SOLUTIONS

1. You can use grid approximation or just the Beta distribution directly. I'll show both. The grad approximation can reuse the code from the chapter. But I'll rewrite it a bit so the function accepts the counts W and L instead:

```
compute_posterior <- function( W , L , poss=c(0,0.25,0.5,0.75,1) ) {
    ways <- sapply( poss , function(q) q^W * (1-q)^L )
    post <- ways/sum(ways)
    data.frame( poss , ways , post=round(post,3) )
}
compute_posterior( 3 , 11 , poss=seq(from=0,to=1,len=11) )
```

```
   poss         ways  post
1   0.0 0.000000e+00 0.000
2   0.1 3.138106e-04 0.171
3   0.2 6.871948e-04 0.374
4   0.3 5.338782e-04 0.291
5   0.4 2.321901e-04 0.126
6   0.5 6.103516e-05 0.033
7   0.6 9.059697e-06 0.005
8   0.7 6.076142e-07 0.000
9   0.8 1.048576e-08 0.000
10  0.9 7.290000e-12 0.000
11  1.0 0.000000e+00 0.000
```

Using the Beta distribution means we don't really need to compute the distribution. We have a mathematical expression for it. There's nothing to compute. But you can plot it with:

```
curve( dbeta(x,3+1,11+1) , from=0 , to=1 , xlab="p" )
```

2. Let's sample from the Beta distribution and then simulate globe tosses from those samples:

```
p_samples <- rbeta(1e4,3+1,11+1)
W_sim <- rbinom(1e4,size=5,p=p_samples)
```

I used the rbinom() function, but you could use sample() and then tally the water points. The resulting distribution is approximated by the counts in W_sim. You can view the distribution with:

```
plot(table(W_sim))
```

3 - CHALLENGE. The first insight needed here is to define a sequence for N rather than for p. Then the same code almost works. Only almost because N has a known lower bound—it is at least W. And it has no defined upper bound. The globe could in principle be tossed an infinite number of times. Not practically but mathematically. So our posterior function needs to know how large an N we'd like to consider.

The second insight is that unlike the book example, the sequence of W and L isn't known here. So we have to consider how many different sequences could produce any particular mix of W and L. Luckily the binomial distribution does this for us. I'll make the calculation explicit, but you could just use dbinom().

```
compute_posterior_N <- function( W , p , N_max ) {
    ways <- sapply( W:N_max ,
        function(n) choose(n,W) * p^W * (1-p)^(n-W) )
    post <- ways/sum(ways)
    data.frame( N=W:N_max , ways , post=round(post,3) )
}
compute_posterior_N( W=7 , p=0.7 , N_max=20 )
```

```
1   7 0.082354300 0.058
2   8 0.197650320 0.138
3   9 0.266827932 0.187
4  10 0.266827932 0.187
5  11 0.220133044 0.154
6  12 0.158495792 0.111
7  13 0.103022265 0.072
8  14 0.061813359 0.043
9  15 0.034770014 0.024
10 16 0.018544008 0.013
11 17 0.009457444 0.007
12 18 0.004642745 0.003
13 19 0.002205304 0.002
14 20 0.001017833 0.001
```

Since p is greater than 0.5, we expect most tosses to be W, so the posterior distribution for N assigns most of the probability to values close to the observed $W = 7$. If we make p small, we'll get the opposite:

```
compute_posterior_N( W=7 , p=0.2 , N_max=20 )
```

```
    N        ways  post
1   7 0.000012800 0.000
2   8 0.000081920 0.000
3   9 0.000294912 0.001
4  10 0.000786432 0.004
5  11 0.001730150 0.008
6  12 0.003321889 0.015
7  13 0.005757941 0.027
8  14 0.009212705 0.043
9  15 0.013819057 0.064
10 16 0.019653770 0.091
11 17 0.026729128 0.124
12 18 0.034990858 0.163
13 19 0.044321754 0.206
14 20 0.054549850 0.253
```

If you had any prior information about N, you could add that to the function as well. Just multiply. For example, suppose you recall that you always toss the globe an even number of times. Then we could just zero out the odd numbers and renormalize.