To introduce OpenBUGS, we use an example derived from Ex. 2.3.1 in the textbook. We consider the prevalence of drug use in the transportation industry. We assume that we've tested 100 transportation industry workers for drug use, and we've obtained 15 positive test results.

Independently of these data, a researcher has estimated that 10% of transportation workers use drugs on the job; and the researcher is 95% sure that no more than 25% of transportation workers use drugs on the job.

This is about as simple as models get in WinBUGS / OpenBUGS, and it's a nice bit of code to play around with, so you can get a feel for some of the basic BUGS functions (e.g. the Sample Monitor Tool).

To sample from this model, choose "Specification..." under the Model menu. Then, highlight the model statement below and click "check model". A message should appear at the bottom of the page saying the model is syntactically correct—or giving a (hopefully) helpful error message if it isn't.

```
model{    X ~ dbin( p, n )
          p ~ dbeta( a, b )    }
```

Then highlight the first list below and click "load data", followed by "compile". If everything's working correctly, the two "inits" buttons will become clickable. You can either highlight a new list statement, like the one below, and click "load inits", or you can click "gen inits" and let BUGS pick initial parameter values for you.

```
list(    X = 15,    n = 100,    a = 3.4,    b = 23    )
list(    p = 0.5    )
```

Once your model is set in the system, you can use the tools in the Inference menu to monitor your parameter sample. Today, we'll look at the Sample Monitor Tool, which comes up when you click "Samples..." under that menu.

Finally, to obtain your parameter sample, choose "Update..." under the Model menu. Choose the size of the parameter sample you want and type this number into the "updates" field. Then click the "update" button, and BUGS will create a parameter sample for you.

Be careful to "burn in" your parameter samples before telling BUGS to monitor the parameters. Remember that the values we sample are only coming from the posterior distribution after our stochastic process converges to a stationary distribution. Use the Update Tool to run a small number of updates before setting nodes using the Sample Monitor Tool. Then, after setting your nodes, use the Update Tool to generate more updates. This second run of updates are the values you sample out of the posterior distribution for the nodes you've set.

If we run a burn-in of 1000 iterations and sample 5000 values from the posterior for p, we can learn more about the posterior of p by using the "stats" button on the Sample Monitor

Tool. Your results should look something like this:

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
p	0.1462	0.03182	4.305E-4	0.09004	0.144	0.2155	1001	5000

Now that we've constructed our first model and posterior sample, we consider a slightly harder problem: comparing the drug use rates between transportation workers and (non-pilot) airline industry workers.

For the airline industry, we collect new data on 150 workers and find that 10 of them test positive for drug use. We do not have access to good prior information about drug use in this industry, however, so we choose a flat prior over the space of possible drug use proportions.

NB that all the numbers in these examples are totally made up! Fake news! Do not use these numbers to make unfounded generalizations about actual transportation or airline industry workers.

```
model{    X ~ dbin( p_1, n_1 )
          Y ~ dbin( p_2, n_2 )
          p_1 ~ dbeta( a_1, b_1 )
          p_2 ~ dbeta( a_2, b_2 )
          d_1 <- p_1 - p_2
          d_2 <- p_1 / p_2    }

list(    X = 15,    n_1 = 100,    a_1 = 3.4,    b_1 = 23,
       Y = 10,    n_2 = 150,    a_2 = 1,    b_2 = 1    )
```

Looking at the summary statistics for `p_1` and `p_2` can tell us about the proportion of workers in each occupational category testing positive for drug use. Looking at the summary statistics for `d_1` and `d_2` can tell us the difference and ratio (respectively) between those proportions. Typing an asterisk, '*', in the node dropdown for the Sample Monitoring Tool will allow us to get information on all monitored parameters simultaneously.

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
d_1	0.07313	0.03776	5.102E-4	-4.704E-4	0.07238	0.1499	1001	5000
d_2	2.21	0.8951	0.01226	0.9955	2.031	4.349	1001	5000
p_1	0.1454	0.03076	4.164E-4	0.09063	0.1431	0.2133	1001	5000
p_2	0.07225	0.02128	2.707E-4	0.0365	0.07028	0.1198	1001	5000

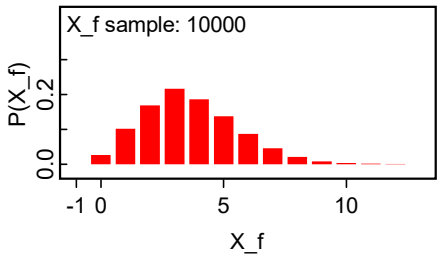
Third, let's consider predicting the number of positive test results we'd see if we used our existing (fake) data and prior information on transportation industry workers. Let's say we

test 25 more transportation workers for drug use. How many workers do we expect to test positive?

```
model{    X ~ dbin( p, n )
          p ~ dbeta( a, b )
          X_f ~ dbin( p, n_f )    }

list(    X = 15,    n = 100,    a = 3.4,    b = 23,    n_f = 25    )
```

In this model, X_f represents the new sample we're taking. p is the same parameter we were estimating with our existing data and prior, and n_f is the sample size for the new sample. BUGS will generate a posterior sample of values for p and then use these posterior values to create samples from the predictive distribution on X_f. We can use the "density" button in the Sample Monitor Tool to get an estimate of the resulting predictive density.



Now let's consider a new binomial example—the problem of defective computers produced by two different companies (we'll call them Hokushiba and Starfruit). We think that, in general, about 3% of computers may be defective. Furthermore, we're almost sure that the proportion of defective computers can't be above 10%. BetaBuster lets us find a prior based on those beliefs, and we wind up with a Beta(2.6, 53.6) distribution.

We gather data on 200 computers from each of Hokushiba and Starfruit, and we find that 11 of the Hokushiba computers are defective and 5 of the Starfruit computers are defective. We're interested in ordering 500 new computers from UNM, so let's look at the corresponding predictive distribution of defective computers for both Hokushiba and Starfruit.

```
model{    Hoku ~ dbin( p1, n )
          Star ~ dbin( p2, n )
          p1 ~ dbeta( a, b )
          p2 ~ dbeta( a, b )
          Hoku_f ~ dbin( p1, n_f )
          Star_f ~ dbin( p2, n_f )
          Check1 <- step( Hoku_f - Star_f )
          Check2 <- step( Hoku_f - (Star_f * 2) )
          Check3 <- Hoku_f - Star_f    }
```

list(Hoku = 11, Star = 5, n = 200, a = 2.6, b = 53.6, n_f = 500)

As before, we can look at the predictive densities for both Hokushiba and Starfruit, but we'd like to consider some other questions as well. First, let's say we want to minimize the number of defective computers we buy. Check1 is an indicator function telling us whenever the number of defective computers from Hokushiba is greater than the number of defective computers from Starfruit—so if we look at the mean of this variable, it estimates the probability that we'd get more defectives from Hokushiba than Starfruit.

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
Check1	0.8754	0.3303	0.005362	0.0	1.0	1.0	1001	5000

But let's take this example a step further. Say a Hokushiba computer costs $500 and a Starfruit computer costs $1000, and we want to minimize how much money we spend on defective computers. Since each Starfruit computer costs 2x as much as a Hokushiba computer, we want to know the probability that we'd get more than twice as many defectives from Hokushiba as we'd get from Starfruit. Check2 lets us look at that probability.

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
Check2	0.4544	0.4979	0.006386	0.0	0.0	1.0	1001	5000

Please feel free to play around with these examples as much as you like as you try to learn how the BUGS system operates!