A crash course to Stan's syntax.

The basic syntax is similar to all "curly bracket" languages, such as C and JavaScript. But assignment and vectorization is similar to R.

```
// This is a comment (but # also works)
# Here are some legal Stan statements
x = x + 1;
x = sqrt( (x + 1) * 3 );
if( x > 10) {
  x = 0;
}
# ";" is necessary after each statement.
```

As opposed to JavaScript, R and python, Stan is statically typed, and there are a lot of types specific to statistical modelling.

```
# Basic types
real x;
int y;
# Vectors are list of real numbers
vector[10] v; # A vector of length 10
# Matrices are tables of real numbers
matrix[2,4] mat; # A 2 by 4 matrix
# Arrays are lists of any data type
int a[4]; # A 4 length array of integers
# Arrays can also be of higher dimension
int a2[4, 3]; # A 4 by 3 array
```

All types can have *constraints*. Constraints are required for variables acting as *parameters*.

```
real mu; # No constraint implies [-∞, ∞]
real<lower=0> sigma; # [0, ∞]
int y<lower=0, upper=1>; # a "boolean"
# Works for vectors and arrays as well
vector<lower=1>[3] v;
int<upper=0> a[4];
# There are many "speciality" data types
simplex[3] p; # A vector of 3 positive
              # reals that sums to 1.
corr_matrix[3] Sigma; # a 3 by 3
                      #correlation matrix
```

A Stan program consists of a number of *blocks*.

```
data { # the required data for the model
  # Declarations ...
}
parameters { # the model's parameters
  # Declarations ...
}
model { # Defines the statistical model
  # Declarations followed by statements ...
}
generated quantities {
  # Declarations followed by statements ...
}
# But there are more block types...
```

Sampling statements define statistical relations between parameters and data.

```
x ~ normal(mu, sigma); # Read as: x is
  # distributed as a normal distribution
  # with mean mu and SD sigma.
# There are many built in distribution.
mu ~ uniform(0, 100);
sigma ~ gamma(2, 2);
x1 ~ student_t(nu, mu, sigma);
lambda ~ exponential(1);
y ~ poisson(lambda);
p ~ beta(a, b);
s ~ binomial(30, p);
```

As in R, many functions are vectorized.
Here assuming **v1** and **v2** are vectors.

```
sum(v1); # Sums all elements in v;
v1 + v2; # The vector of the pairwise sum
         # of v1 and v2
# Assume all elements of v are normal.
v ~ normal(mu, sigma);
# This is equivalent to using the for loop:
for (i in 1:n) {
  v[i] ~ normal(mu, sigma);
}
# Note that “[]” is used to index vectors
and arrays, and that 1 is the first index.
```

A minimal Stan program implementing a binomial model.

```
data {
   int n;
   int x;
}
parameters {
   real<lower=0, upper=1> p;
}
model {
  p ~ uniform(0, 1);
  x ~ binomial(n, p);
}
```

Running a Stan program is usually done from another language such as Python or R.

(Here assuming `model_string` contains the model from the last slide.)

```
library(rstan)
data_list <- list(n = 30, x = 10)
s <- stan(model_code = model_string,
          data = data_list)
```

```
import pystan
data_list = dict(n = 30, x = 10)
s = pystan.stan(model_code = model_string,
                data = data_list)
```