Download and go through "R: A self-learn tutorial" from
http://www.nceas.ucsb.edu/files/scicomp/Dloads/RProgramming/BestFirstRTutorial.pdf

Turn in your R output for the following.

Section	Exercises
2. Objects and Arithmetic	1, 2
3. Summaries and Subscripting	1
4. Matrices	1 (a,b, & e), 2
7. Statistical Computation and Simulation	1, 3
8. Graphics	1, 3

===

 # Mathematical operations
2+2
(4*3)/2

 # Built-in functions
sqrt(144)
log(100)
exp(4.60517)

 # Assignment of values
x <- 2.4
y <- sqrt(16)
z <- x*y
z

 # Sequences
x <- 1:100
x
# seq(from, to, by)
# e.g., to make a sequence of numbers from 0 to 100 in increments of 2
x <- seq(0,100,2)
x

 # vectors
S <- c(0,12,14,84,312)
S
mean(S)
length(S)
2*S
S[3]
S[2:4]

In R, operations on vectors are often applied to each element of the vector.
So when x holds the vector (4, 2, 6), the command x-2 subtracts 2 from each element of the vector, yielding (2, 0, 4).

 # matrices

```
m1 <- c(0.2, 1.2, 1.4, 0)
m2 <- c(0.3, 0, 0, 0)
m3 <- c(0, 0.4, 0, 0)
matrix1 <- rbind(m1, m2, m3)
matrix1
matrix1[3,2]
matrix1[2, ]
matrix1[ , 2]
```

 # data frames

```
# "trees" is included in the base packagae
str(trees)
summary(trees)
trees$Height
```

 # graphics

```
plot(trees$Height ~ trees$Girth)
with(trees, plot(Height ~ Girth))
attach(trees)
plot(Girth, Height)
```

 # for loop

```
steps <- 8;             # number of time steps to be simulated
lambda <- 1.08;
N <- numeric(steps+1);   # creates an empty vector; filled with 0's
N[1] <- 5;        # the initial population size is assigned to the 1st element

for(i in 1:steps){
  N[i+1] <- N[i]*lambda
}
N

plot(N)
plot(N, type = 'l')
```

 # statistical distributions

```
rnorm(10)  # random draw of 10 values from a normal distribution; mean = sd = 1
rnorm(10, 2, 0.8)  # 10 values with mean = 2, sd = 0.8

# density at x = 1.3 of the normal distribution with mean = 2.1 and sd = 0.9
dnorm(1.3, 2.1, 0.9)

#  cumulative probability of the normal distribution (with mean = 2.1 and sd = 0.9)
#    at x = 2.5; that is, the proportion of the normal distribution that lies below x = 2.5
pnorm(2.5, 2.1, 0.9)
```

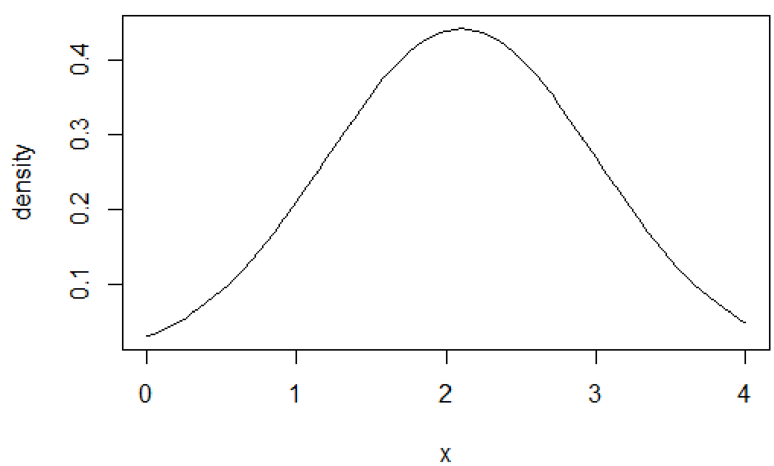
The result, 0.6716394, means that 67.16% of the normal distribution lies below x = 2.5.

For the same mean and s.d., the probability that x lies between -1.1 and 1.8 is given by:
pnorm(1.8, 2.1, 0.9) - pnorm(-1.1, 2.1, 0.9)

Section 7, exercise 7, asks you to find the values of density, distribution, and quantile functions for a normal distribution. In R, these are called dnorm, pnorm, and qnorm respectively. Plotting examples may help you to understand the functions. Consider a normal distribution with a mean of 2.1 and a standard deviation of 0.9

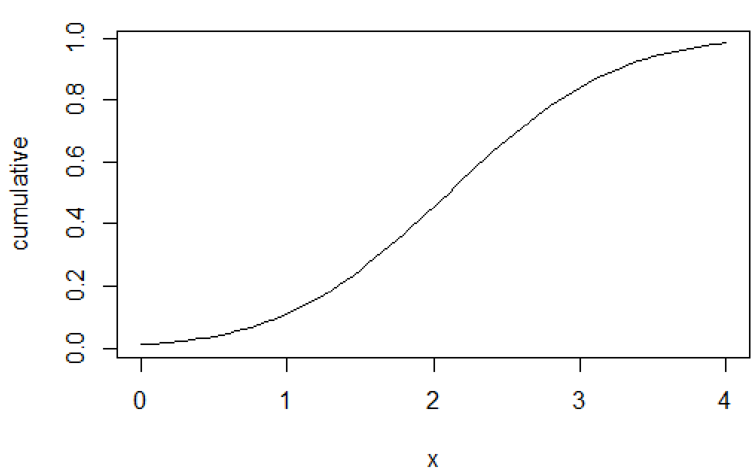
Here is a way to plot the density function dnorm from x = 0 to x = 4.

```
# generate a sequence of closely spaced values from 0 to 4
x <- seq(0, 4, 0.05)
# find the density at each value of x; save the results in an object called “normdensity”.
normdensity <- dnorm(x, 2, 0.25)
# plot the density versus x; type = “l” produces a line graph
plot(normdensity ~ x, type = "l", ylab = "density")
```



Similarly, to make a plot of the distribution function (cumulative probability):

```
cumulative_normal <- pnorm(x, 2.1, 0.9)
plot(cumulative_normal ~ x, type = "l", ylab = "cumulative")
```



```
# Stochastic population growth
steps <- 25;                    # number of time steps to be simulated
N <- numeric(steps+1);          # creates an empty vector; filled with 0's
lambda <- numeric(steps);       # creates an empty vector
N[1] <- 10;                     # the initial pop. density is assigned to the 1st element

for(i in 1:steps){
  lambda[i] <- rnorm(1, 1.05, 0.1)      # draw a number from the normal distribution
  N[i+1] <- N[i]*lambda[i]              # calculate the new pouplation density
}
N                               # show the values of N
lambda                          # show the values of lambda

plot(N, type = 'l')             # plot the population trajectory
```