Introduction to R

**Useful functions:**

seq(),c(): Two functions that can be used to create vectors.

matrix(): Creates a 2-dimensional object

length(): Returns the length of a vector or list

rnorm(): Generates normal random numbers. See also runif(), rpois() . . .

sum(): Returns the sum of a numeric object (vector, matrix etc.)  
apply(): Apply some function to the rows or columns of a matrix

read.csv(): Read a csv file into a data frame

head(), tail(), dim(), str(): Useful for exploring the structure of a data frame

which(): Which elements of a logical vector or array are TRUE?

**Some syntax notes:**

for( *variable* in *vector* )

{

functions

}

Details: Here *variable* is the index that will start at the first value of *sequence* and continue until the last value of *sequence*. *Sequence* will often be defined using the : operator, but could be any sequence, including one from the function seq(). Starting with the first and continuing through each element in *sequence*, the functions within the curly brackets will be executed.

while( *condition* )

{

functions

}

Details: *Condition* represents some logical test that is either true or false. For example X > 5, or length(X) > 0, or X == Y (note the ==. This is different from =). This will execute the functions in the curly brackets for as long as the condition is true. If you program this wrong, the loop can continue forever!

if( *condition* )

{

functions

}

Details: if *condition* is true, the operations between the curly brackets are executed.

if ( *condition* ) { functions } else { other functions }

Details: As with an if statement, the first set of functions are executed if the *condition* is true, otherwise the functions following else are executed.

Part 1: Creating and indexing numeric variables

**1**. Like many functions in R, seq() is flexible and can work in several different ways. It is worth understanding how and why it works exactly the way it does. Try the following calls:

seq()

seq(0,5)

seq(from = 0, to = 5)

seq(to = 5)

seq(to = 5,by = 0.5)

seq(from = 10, to = 1, length.out = 10)

seq(from = 5, to = 1, by = -2)

seq(from = 5, to = 1, by = 2)

Do the returns of all of these function calls make sense?

Note that you can pass arguments to a function in R either by name (e.g. to = 5, by = 0.1), or just by putting things in the right order. The function seq expects you to give it arguments in the order (*from*, *to*, *by*). You can find that out by checking the help file on seq. If in doubt, it is safest to specify arguments by name, in which case any order is fine. For example:

seq(4,8,1)

seq(by = 1, to = 8, from = 4)

seq(to = 8, by = 1, from = 4)

are interpreted identically by R.

One final note: if you are interested in generating an integer sequence, it is often fastest to use an *operator*, a symbol reserved for a special, commonly used task. Here, we could generate the above sequence using the : operator

4:8

Other common operators include +, -, \*, /, ^, among others. Try them out!

**2**. Use seq() and c() to create a vector called “a” that contains the integers from 1 to 10, followed by the even integers from 20 to 30, followed by the multiples of 5 from 150 to 100 (counting down). What is the 14th element of a? Use a single line of R code to find the 2nd, 23rd and 10th elements of a, in that order. Use length() to find the length of a.

**3**. Create a matrix (using matrix() ) called A that has 12 rows and 8 columns, where each element in A is a different randomly generated number from a normal distribution with mean of 3 and standard deviation of 2 (rnorm() generates normal random numbers).

Create a new object b and assign in the value from the 5th row and 6th column of A. Create a new object called A2 which is just the 3rd,4th and 5th row of A. Note that once they are set, b and A2 are independent of A. They will keep their values as defined, even if the elements of A later change.

Part 2: Loading and manipulating data

**4**. Use read.csv() with the as.is=T option set to read in an example data frame with species abundances for 10 sites (“Abundance.csv”), and call the data frame “Abundance”. Use head(), names() and str() to explore the structure of the data frame.

**5**. Use the function apply() to calculate the average abundance of each species across all 10 sites. Note that we are working with a data frame now, so not all the columns are numeric. The species name column will not enjoy having numeric operations performed on it, so you will need to exclude that column in some way.

**6a**. Create a new matrix called “Presence” where each element of Presence is 1 if the corresponding value of Abundance > 0, otherwise 0. There are lots of ways to solve this, and it will help to think about it a bit before you start typing!

**b**. Using the Presence matrix you created and the apply() function, calculate the species richness of each site.

**c.** Write the Presence matrix to a .csv file.

A challenge if there is time

**Extra 1**. Write a for loop to generate the first 100 numbers of the Fibonacci sequence. The Fibonacci sequence is (1,1,2,3,5,8,13…) where the first two numbers are 1, and each following number is equal to the sum of the previous two. What is the 40th number in the sequence? (The answer should be 102334155)