Week 1 Exercise

Z620: Quantitative Biodiversity, Indiana University

November 4, 2014

In this exercise, we provide an introduction to some of the basic features of the R computing environment. We emphasize calcuations, data types, and simple commands that will be useful for you during the course.

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

RETRIEVING AND SETTING YOUR WORKING DIRECTORY

```
getwd()
```

```
## [1] "/Users/lennonj/GitHub/Quantitative_Biodiversity/Assignments/Week1"
```

```
setwd("/Users/lennonj/GitHub/Quantitative_Biodiversity/Assignments/Week1")
```

USING R AS A CALCULATOR

addition

```
1+3
```

```
## [1] 4
```

subtraction

```
3-1
```

```
## [1] 2
```

multiplication (with exponent)

```
3*10^2
```

```
## [1] 300
```

division (using a built-in constant)

```
10/pi
```

```
## [1] 3.183
```

trigonometry with a simple built-in *function* (i.e., 'sin') and *argument* (i.e., '4')

```
sin(4)
```

```
## [1] -0.7568
```

logarithms (another example of function and argument)

```
log10(100)
```

```
## [1] 2
```

```
log(100)
```

```
## [1] 4.605
```

DEFINING VARIABLES

In R, you will often find it useful and necessary to assign values to a variable. Generally speaking, it's best to use '<-' rather than '=' as an assignment operator.

```
a <- 10
b <- a + 20
```

What is the value of b?

```
a <- 200
```

Now what is the value of b? Can you explain? Fix? It can help to examine variables with the following function

```
ls()
```

```
## [1] "a" "b"
```

You can clear variables from R memory with following function (example of nested function)

```
rm(list=ls())
```

You can also examine variables in the Environment windwow of R Studio. By clikcing 'clear' in this window, you can erase variables from memory

WORKING WITH SCALARS, VECTORS, AND MATRICES

Create a *scalar* by assigning a numeric value to a character

```
w <- 5
```

A *vector* (or array) is a one-dimensional row of numeric values. You can create a vector in R like this:

```
x <- c(2,3,6,w,w+7, 12,14)
```

What is the function 'c'? The 'help' function is your friend.

```
help(c)
```

What happens when you multiply a vector by a scalar?

```
y <- w*x
```

What happens when you multiply two vectors?

```
z <- x*y
```

Here is how you reference an element in a vector

```
z[2]
```

```
## [1] 45
```

Here is how you reference multiple elements in a vector

```
z[2:5]
```

```
## [1]  45 180 125 720
```

Here is how you can change the value of an element in a vector

```
z[2]=583
```

It's pretty easy to perform summary statistics on a vector using built-in fuctions

```
max(z)
```

```
## [1] 980
```

```
min(z)
```

```
## [1] 20
```

```
sum(z)
```

```
## [1] 3328
```

```
mean(z)
```

```
## [1] 475.4
```

```
median(z)
```

```
## [1] 583
```

```
var(z)
```

```
## [1] 133881
```

```
sd(z)
```

```
## [1] 365.9
```

What happens when you take the standard error of the mean (sem) of z? Sometimes you need to make your own functions:

```
sem <- function(x){
  sd(x)/sqrt(length(x))
  }
```

Often, datasets have missing values (designated as 'NA' in R)

```
i <- c(2,3,9,NA,120,33,7,44.5)
```

What happens when you apply your sem function to vector i? One solution is to tell R to remove NA from the dataset:

```
sum(i,na.rm=TRUE)
```

```
## [1] 218.5
```

There are three common ways to create a matrix (two dimensional vectors) in R. **Approach 1** is to combine (or concatenate) two or more vectors. Let's start by creating a vector using a new function 'rnorm'

```
j <- c(rnorm(length(z),mean=z))
```

What does the rnorm function do? What are arguments doing? Now we will use the function 'cbind' to create a matrix

```
k <- cbind(z,j)
```

Use the 'help' function to learn about cbind Use the 'dim' function to describe the matrix you just created

Approach 2 to making a matrix is to use the matrix function:

```
l <- matrix(c(2,4,3,1,5,7),nrow=3,ncol=2)
```

Approach 3 to making a matrix is to import or 'load' a dataset from your working directory (or elsewhere)

```
m <- as.matrix(read.table("matrix.txt",sep="\t",header=FALSE))
```

Often, when handling datasets, we want to be able to transpose a matrix. This is easy in R:

```
n <- t(m)
```

Also, you will find that you need to subset data in a matrix:

For example, maybe you want to take first three rows of a matrix:

```
n <- m[1:3,]
```

Or maybe you want the first two columns of a matrix:

```
n <- m[,1:2]
```

Or perhaps you want non-sequential columns of a matrix:

```
n <- m[,c(1:2,5)]
```

Other things to address: >ordering/sorting >logic >subsetting by characters > plotting