Elizabeth Murray, 2016
ENTOM 4610

For new R users, practice working with the data structures:

Here, you will explore some of the inputs, outputs, and options of the various types of data structures that you can use in R. You will get a sense of how to manipulate data.

To start, we'll enter a string of numbers and assign it to an object.

First, use the "?" to get the help page on the function 'scan'. Type this entry (NOT including the "> ").

```
> ? scan
```
take a look at the options and arguments for the scan function
None of the arguments are required. How can you tell?

it's a good idea to use object names that do not overlap with existing functions
object names cannot start with a numerical value (e.g. "1list")

Type this entry. Remember the '<-' operator assigns values to an object. Create the object 'my.vector'.

```
> my.vector <- scan()
```

After you enter this first line, you'll see that R is waiting for more input, as signified by the "1: ".
Now, enter some numerical values and press enter. R will wait for more input (and "11: " is displayed), but just press enter again to finish. You'll get a message that the items were read successfully.

```
1: 1 3 5 7 9 2 4 6 8 0
11:
Read 10 items
```

You can view the mode and structure of the vector by using these base functions.
The vector is comprised of the numbers you entered via the scan function (each an element).

```
> mode(my.vector)
[1] "numeric"
> str(my.vector)
 num [1:10] 1 3 5 7 9 2 4 6 8 0
```

We will ask which value(s) is equal to 3 (or, can use other operators such as > or <)

```
> which (my.vector == 3)
[1] 2
```
the '2' indicates the 2nd element of the vector

Access one or more of the elements:

```
> my.vector[[5]]
[1] 9
```
notice, you're accessing the position of the 5th element, NOT the numerical value of 5

```
> my.vector[1:3]
[1] 1 3 5
```
extract a subset by using single brackets

You can assign selected numbers to a new object (they will remain also in the my.vector object)

```
> subset <- my.vector[1:3]
> subset
[1] 1 3 5
```

Let's make a new vector of the same length and combine the two together

```
> vector2 <- c(2,4,6,8,0,1,3,5,7,9)
> vector2
 [1] 2 4 6 8 0 1 3 5 7 9
```

This time instead of using scan, I used the concatenate function to combine these objects as a vector. You have to put commas in between each of the variables. With scan(), you just need to put spaces.

We can add each of the corresponding elements:

```
> my.vector+vector2
 [1]  3  7 11 15  9  3  7 11 15  9
```

We can also put one right after another:

```
> c(my.vector, vector2)
[1] 1 3 5 7 9 2 4 6 8 0 2 4 6 8 0 1 3 5 7 9
```

What if we have these two vectors, and what we really want is to combine them into a matrix?
A matrix is a vector with the added attribute 'dimensions', or, dim.

```
> my.matrix <- cbind(my.vector, vector2)
> my.matrix
      my.vector vector2
 [1,]         1       2
 [2,]         3       4
 [3,]         5       6
 [4,]         7       8
 [5,]         9       0
 [6,]         2       1
 [7,]         4       3
 [8,]         6       5
 [9,]         8       7
[10,]         0       9
```

in this case, the dimensions of our matrix are 10 X 2 (check by using `> dim(my.matrix)`)

Now, we have to use both the ROW and COLUMN [r,c] location to access an element(s) of the list:

```
> my.matrix [[9,1]]
[1] 8
> my.matrix [1:3, 1:2]
     my.vector vector2
[1,]         1       2
[2,]         3       4
[3,]         5       6
```

remember, you could also assign these certain accessed variables to their own objects

Our object "my.matrix" is a matrix. We can check this by verifying the class.

```
 > class(my.matrix)
[1] "matrix"
```

A matrix is different than a data frame in that all of the variables in a matrix are the same type (character, numerical, etc.). It's like a glorified vector, while a data frame is a type of list and can be made from vectors and lists. You'll probably be using these often.

We can convert matrices to data frames. This is useful in some R phylogenetic package functions that require the data to be in data.frame format.

```
> my.df <- as.data.frame(my.matrix)
> my.df
   my.vector vector2
1          1       2
2          3       4
3          5       6
4          7       8
5          9       0
6          2       1
7          4       3
8          6       5
9          8       7
10         0       9
```

```
> class(my.df)
[1] "data.frame"
```

How are these two different in their structures?

```
> str(my.matrix)
```

```
> str(my.df)
```

Try accessing the 2nd column of the matrix and of the data.frame using the operator **$**. What happens?
Column 2 of the matrix?

```
> my.matrix$vector2
```

Column 2 of the data frame?

```
> my.df$vector2
```

The operator **$** only works for the data frame. There is another way to access the second column -- remember the brackets? You can use this for the matrix (and it works for the data frame too):

```
> my.matrix[,2]
```

using no integer but putting in a comma for the row selection means "use ALL rows"

```
> my.matrix[,2, drop=F]
```

how does this output look different? You have told it not to turn the extraction into a vector.

Some other handy related functions to check out:

```
as.character()
names()
attr()
paste()
```

```
fix()
is.vector()
row.names(); column.names()
apply(); lapply()
```