Introduction to Using R

R is an object oriented programming language. We are NOT going to learn to write programs in R. Rather, we will apply R functions written by others to our data sets in order to analyze the data and then we will interpret the output from these functions. In order to understand what R is doing for us, there are a few fundamentals about the language and how it operates that we should understand.

Prompts:

When R is waiting for us to tell it what to do, it begins the line with

```
>
```

If we give it an incomplete command and it can not finish the task requested it provides

```
+
```

To get out of R we use the command

```
> q()
```

R as a Calculator:

At the prompt, we enter the expression that we want evaluated and when we hit enter, it will compute the result for us. For Example:

```
>  4+6
```

Use * for multiply.
Use ^ for raised to the power of.
Use parentheses to ensure that it understands what you are trying to compute.
The order of doing arithmetic operations is (left [done first] to right [done last])

```
   ^   /   *   -   +
```

Built-in Functions:

```
pi

exp(3)  ## provides the cube of e

log(1.4)  ## provides the natural logarithm of the number 1.4

log10(1.4)  ## provides the log to the base of 10

sqrt(16)   ##  provides the square root of 16
```

Assignment Statements:

Just like in algebra, we often want to store a computation under some variable name. The result is assigned to a variable with the symbols < - which is **formed by the "less than" symbol followed immediately by a hyphen.**

```
>  x < - 2.5
```

When you want to know what is in a variable simply ask by typing the variable name.

```
> x
```

We can store a computation under a new variable name or change the current value in an old variable.

```
> y < - 3*exp(x)
> x < - 3*exp(x)
```

Variable (Object) Names:

Certain variable names are reserved for particular purposes. Some reserved symbols are:

c q t C D F I T

You should create variable names that show the meaning of what is computed. You may use a period to extend its description. But do not begin a variable name with a period or a number. Variable names are case (upper/lower) sensitive. Examples:

mean.height brand.A abbott.costello

Creating a Vector:

To create a vector we "concatenate" a list of numbers together to form a vector.

```
> x < - c(1, 6, 4, 10, -2)
```

Once we have a vector of numbers we can apply certain built-in functions to them to get useful summaries. For example:

```
> sum(x)     ## sums the values in the vector

> length(x)     ## produces the number of values in the vector, ie its length

> mean(x)     ## the average (mean)

> var(x)     ## the sample variance of the values in the vector (has n-1 in denominator)

> sd(x)     ## the sample standard deviation of the values in the vector  (square root of the sample variance)

> max(x)     ## the largest value in the vector

> min(x)     ## the smallest number in the vector

> median(x)     ## the sample median

> y < - sort(x)    ## the values arranged in ascending order
```

Indexing :

The elements in a vector are indexed, indicating which is the first element, second etc. But only the first index value is printed. If the vector is too long for one line of printing then more lines are used. It does print the index of the first element printed in each line.
For example:

```
>  x < -  c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
>  x
```

Creating a matrix:

Suppose you were interested in the matrix

$$\begin{array}{ccc} 6 & 2 & 10 \\ 1 & 3 & -2 \end{array}$$

You could form the rows and then bind them together to form the matrix.

```
>  xr1 < -  c( 6, 2, 10)
>  xr2 < -  c(1, 3, -2)
>  x < -  rbind(xr1, xr2)  ## binds the vectors into rows of a matrix

>  y < -  cbind(xr1, xr2)  ## binds the same vectors into columns of a matrix

>  t(x)   ## this is the transpose of the matrix x
```

The elements in a matrix are indexed by their (Row, Column) positions.

```
>  x
>  y
```

Creating a Data Frame:

Data :

Height	GPA
66	3.80
62	3.78
63	3.88
70	3.72
74	3.69

```
>  student.ht  < -  c( 66, 62, 63, 70, 74)
>  student.gpa  < -  c( 3.80, 3.78, 3.88, 3.72, 3.69)
>  student.data  < -  data.frame(student.ht, student.gpa)
>  student.data

>  plot(student.ht, student.gpa)
```

Reading in a Data Set:

One of the major hurdles in using R in this class is getting a data set into our active workspace in R.
The data sets for exercises will be posted on the class website as a text file. When connected to the internet, you can read in data with the following commands:

```
 > [you-name-data] <- read.table(file="[internet-address-of-data]", col.names=c("[col1-name]","[col2-name]"))
```

Example: (Note that the square brackets shown above [ ] should not be included.)

```
 >  datit <-  read.table(file="http://www.stat.ufl.edu/~rrandles/sta4210/Rclassnotes/data/classdata.txt",
col.names=c("ht","wt"))

 > attach(datit)       ## Be sure to include this attach command.
```

Fitting the Regression Line:

Having a data set in the active workspace (suppose it is stored as student.data), we construct the regression line as follows:

```
>  reg.fit  < -  lm(student.gpa~student.ht)
>  reg.fit
> summary(reg.fit)

> plot(student.ht,student.gpa)
> abline(lm(student.gpa~student.ht))
```

Help:

The are facts stored about each built-in function which you can examine. But they are frequently rather technical in nature.

```
>   help(min)

>  help(lm)
```

To exit a help menu, type q