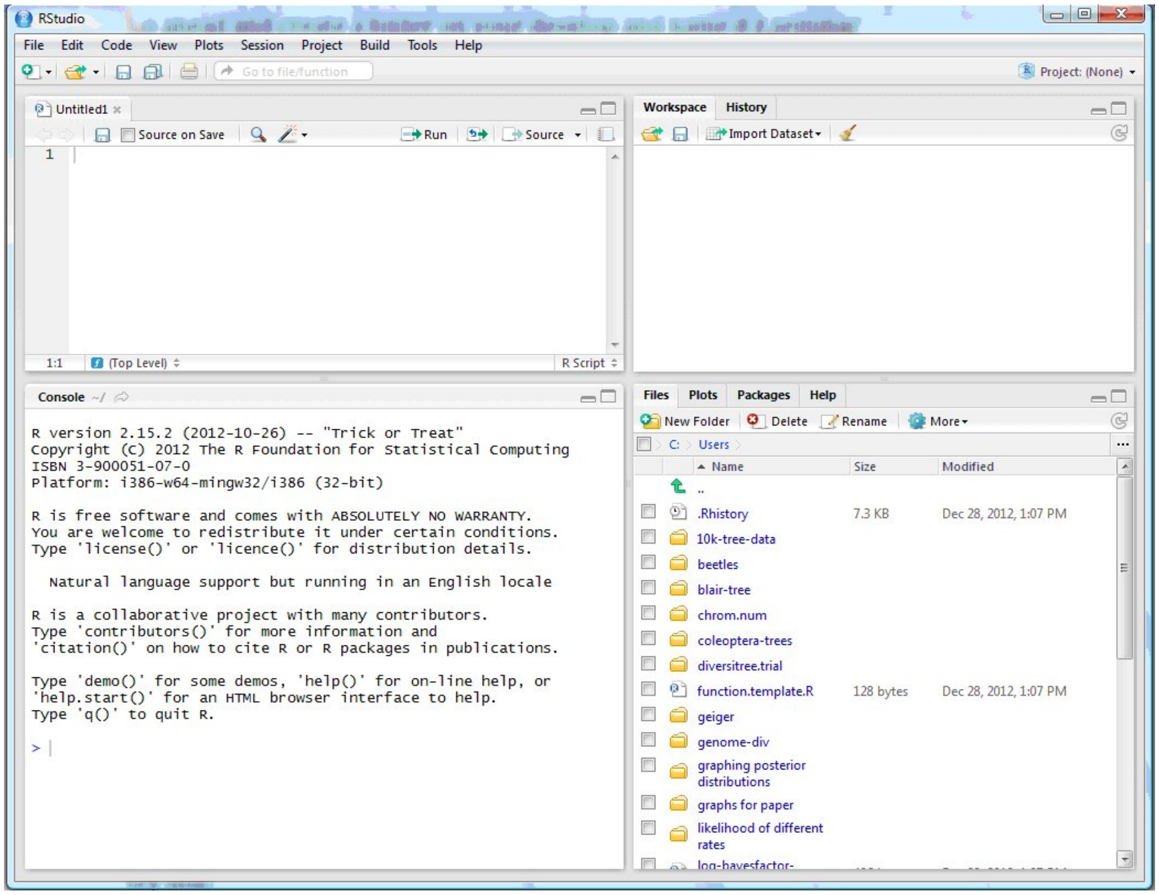
Introduction to R

The goal of our first meeting is to make sure that you are all comfortable with the most basic operations in R.

Go ahead and create a folder on your desktop and name it r.seminar

Now start RStudio and open a new R script (File > New > R Script). Your screen should look like this:



```
R version 2.15.2 (2012-10-26) -- "Trick or Treat"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Name	Size	Modified
..		
.Rhistory	7.3 KB	Dec 28, 2012, 1:07 PM
10k-tree-data		
beetles		
blair-tree		
chrom.num		
coleoptera-trees		
diversitree.trial		
function.template.R	128 bytes	Dec 28, 2012, 1:07 PM
geiger		
genome-div		
graphing posterior distributions		
graphs for paper		
likelihood of different rates		

When you are using R you have a "working directory" this is the folder on your computer that R is going to use when you save or open files. To set the working directory:

Browse with the lower right window in RStudio to your r.seminar folder on your desktop then click:
Windows: Session > Set Working Directory > To files pane location
Mac: Tools > Set Working Directory > To files pane location
Linux: Tools > Set Working Directory > To files pane location

Try out the console and the script editor

The console is in the bottom left quadrant of RStudio you can enter simple commands here. For more complex work that you want to keep track of open an R script and enter your commands in the upper left quadrant.

Oversized calculator

At its most basic, the console is a bloated calculator. The basic operations are
+, -, *, /

for add, subtract, multiply, divide. Familiar calculator functions also work on the command line. For example, enter

```
4^2
log(2)
```

R also knows the order of operations so you should get the correct answer when you try this:

```
4^2*2+1
```

1. Try the calculator out to get a feel for this basic application of R and the style of the output. Try log and a few other functions (sqrt, abs, etc).
2. R allows you to store numbers and characters to variables called vectors, which are a type of "object" in R. For example, to assign the number "3" to a variable "x", use
   ```
   x <- 3
   ```
   The assign symbol is a < followed by a dash -, with no space between. Try assigning a single number to a named variable.
3. R can also assign character data (enter using double quotes) to named variables. Try entering
   ```
   z <- "Wake up Neo"
   ```
4. The upper right quadrant of RStudio will show you all of the objects that are currently being held in memory by R.
5. Assign a number to the variable "y". Then watch what happens when you type an operation, such as”
   ```
   x * y
   ```
   Finally, you can also store the result in a third variable.
   ```
   z <- x * y
   ```
   To print the contents of z, just enter the name on the command line, or enter
   `print(z)`
6. The calculator will also give a TRUE or FALSE response to a logical operation. Try one or more variations of the following examples on the command line to see the outcome. The logic operators are:
   ==, >=, <=, <, >, !=
   Try a few statements out like these

```
2 + 3 == 4
3 >= 2
"a" > "a"
"Hi" != "hi"
```

Vectors

Vectors in R are used to represent variables. R can assign sets of either numbers or a set of characters to named variables using the "c" function (for concatenate). (R treats single numbers or characters as vectors too, having just one element.) A vector may not contain both numbers and characters. Each element assigned to a vector will have an index beginning with 1 and ending with the number elements in the vector. Try creating this vector:

```
x <-  c(1,2,333,65,45,-88)
```

1. Assign a set of 10 numbers to a variable x. Make sure it Includes some positive and some negative numbers. To see the contents afterward, enter x on the command line, or enter print(x). Is it really a vector? Enter is.vector(x) to confirm.
2. Use an integer in square brackets to access the fifth element of your vector x.
   ```
   x[5]
   ```
   Try this out.
   ```
   x[1:3]       # 1:3 is a shortcut for c(1,2,3)
   x[c(2,4,9)]
   ```
   Print the 3rd and 6th elements of x with a single command.
   Create a vector y that has the numbers 1,2,3 in it. Then try this
   ```
   x[y]
   ```
3. Some functions of vectors yield integer results and so can be used as indices too. For example, enter the function
   ```
   length(x)
   ```
   Since the result is an integer, it is ok to use as follows,
   ```
   x[length(x)]
   ```
   The beauty of this construction is that it will always give the last element of a vector x no matter how many elements x contains.
4. Logical operations can also be used to generate indicators. First, enter the following command and compare with the contents of x,
   ```
   x > 0
   ```
   Now enter
   ```
   x[x > 0]
   ```
   Try this yourself: print all elements of x that are non-negative.
   The "which" command will identify the elements corresponding to TRUE. For example, try the following and compare with your vector x.
   ```
   which(x > 0)
   ```
5. Indicators can be used to change individual elements of the vector x. For example, to change the fifth element of x to 0,
   ```
   x[5] <- 0
   ```
   Try this yourself. Change the last value of your x vector to a different number.
   Change the 2nd, 6th, and 10th values of x all to 3 new numbers with a single command.

6. Missing values in R are indicated by NA (without quotes). Try changing the 2nd value of x to a missing value. Print x to see the result.
7. R can be used as a calculator for arrays of numbers too. To see this, create a second numerical vector y of the same length as x. (e.g.: `y <- 1:10`) Now try out a few ordinary mathematical operations on the whole vectors of numbers,
   ```
   z <- x * y
   z <- y - 2 * x
   ```
   Examine the results to see how R behaves. It executes the operation on the first elements of x and y, then on the corresponding second elements, and so on. Each result is stored in the corresponding element of z. Logical operations are the same,
   ```
   z <- x >= y
   z <- x[ abs(x) < abs(y)]
   ```

   What does R do if the two vectors are not the same length? The answer is that the elements in the shorter vector are "recycled", starting from the beginning.
   ```
   v <- c(1,2)
   w <- 1:20
   z <- v * x
   ```
8. A data frame is one common format for data in R. It is basically a collection of vectors that are the same length. We will be using the command "data.frame" to create our data frame. To learn more about how data.frame works you can use the help command just type `help(data.frame)` in the console.

   Make a data frame called mydata from the two vectors, x and y. Print mydata on the screen to view the result. If all looks good, delete the vectors x and y from the R environment (use the command "rm"). They are now stored only in the data frame. Type names(mydata) to see the names of the stored variables.

   We use square brackets again to access elements of the data frame. e.g. mydata[4,1] would give you the fourth row of the first column. If you want all column or rows just leave it blank: mydata[,1] would give all values for x. Try assigning all of the values for y to a vector.

9. Vector functions applied to data frames may give unexpected results -- data frames are not vectors. For example, length(mydata) won't give you the same answer as length(x) or length(y). But you can still access each of the original vectors using mydata$x and mydata$y. Try printing one of them. All the usual vector functions and operations can be used on the variables in the data frame. We'll do more with data frames below.

Exercise 1: Analyze vector of data: finch body weight

We are going to use some data on Darwin's finches just to start experimenting with some basic functions. This data comes from Hau and Wikelski (2000 Encyclopedia of Life Sciences). The data consists of species names, the number island they are found on and the weight of the birds. Download the csv file from the course website and save it in your r.seminar folder. We will read this data in as a data frame using the "read.csv" command:

```
mydata <- read.csv("finch.csv")
```

1. Subset out the values for body weight and store them in a vector. Afterward, check that you have created a vector and the number of observations stored in the vector.
2. Apply the "hist" command to the vector and observe the result (a histogram).
3. The measurements in this dataset are in ounces. The standard international unit though is grams (1 gram = .035 ounces). Transform the finch data so that it is in units of grams and save this back to your same vector.
4. Calculate the sample mean WITHOUT using the function "mean" (i.e., use other functions instead).
5. Ok, try the function "mean" and compare your answer.
6. Calculate the sample standard deviation in weight WITHOUT using the functions "sd" or "var". Then calculate using "sd" to compare your answer.
7. Sort the observations using the "sort" command.
8. Calculate the median weight. When there are an even number of observations (as in the present case), the population median is most simply estimated as the average of the two middle measurements in the sample.
9. Calculate the standard error of the mean weight.

Missing data

Missing data in R are indicated by NA (without quotes). Many functions for vectors, such as sum and mean, will return a value of NA if the data vector you used contained at least one missing value. Overcoming this usually involves modifying a function option to instruct R to remove the offending points before doing the calculation. Call up the help for a function like mean to see how to do this.

1. Use the "c" function to add a single new measurement to the finch vector created in the previous section (i.e., increase its length by one) but have the new observation be missing, as though the weight measurement on a 15th finch species was lost.
2. Check the length of this revised vector, according to R.
3. What is the sample mean of the measurements in the new vector, according to R? Use a method that does not involve you directly removing the offending point yourself.