R, RStudio, and the web

R is a statistical programming language. R is installed on the computers in the engineering lab and you can install it yourself on your own computer. Download it from here [http://www.r-project.org/](http://www.r-project.org/). RStudio is a user interface (IDE or integrated development environment) for R. You can also install RStudio on your own computer. Download it from here [http://www.rstudio.com/ide/](http://www.rstudio.com/ide/). We will actually use RStudio over the web. To get there, go to [http://dahl.calvin.edu:8787/auth-sign-in](http://dahl.calvin.edu:8787/auth-sign-in) from any webbrowser. The machine that you are accessing is dahl, a supercomputer operated by the computer science department. Your username and password are both your Calvin id (e.g., abc23). Notice that RStudio has four panes. The Console is the pane in which we actually enter R commands. We’ll mostly work in it today.

**Arithmetic**

R does arithmetic of course and you can use it as a calculator. For example

```r
> 2 + 3
[1] 5
```

Notice the > is a prompt that says that R is ready for input. Note also the [1] in the output. That says the next number is the first number in the output (not incredibly useful at this point).

Try some arithmetic, using the various arithmetic operations, parentheses, and simple functions. For example, try things like the following. (You might have some guesses as to which functions should be available!)

```r
> 37 * 43
> 2^4
> 1 + 2 * 3
> sin(10)
```

**Variables**

Variables are used to save objects for later use. For example we can save and then use the results of computations. Entering the name of a variable is an implicit command to print what is currently saved in the variable. Notice the peculiar fourth line. It asks R to add 23 to whatever is saved in y and then save the result in y.

```r
> y = 2 + 3
> y
[1] 5
```
Vectors

A vector is an ordered list of objects. A vector has a length and a mode (numeric, character, or logical). We will use vectors to store data. Usually we will not enter the values of vector into R by hand but we will read stored data into R. But short vectors can be entered by hand.

```r
> y = c(1, 2, 3, 9, 8, 7)
> y
[1] 1 2 3 9 8 7
```

The object `y` defined above is a vector and R "knows" that. In most instances, R performs operations on vectors in a natural way. Here are some examples.

```r
> y + 10
[1] 11 12 13 19 18 17
> x = y + 1
> x
[1] 2 3 4 10 9 8
> x + y
[1] 3 5 7 19 17 15
> log(y)
[1] 0.000 0.693 1.099 2.197 2.079 1.946
```
Sometimes it will be necessary to access the pieces of a vector. The vector y has 6 elements.

```r
> y[3]
[1] 3
> y[6]
[1] 7
> x = 5
> y[x]
[1] 8
> y[7]
[1] NA
```

The last result, NA, is read as "Not Available" and denotes a missing value. A value might be missing for a number of reasons – in this case the vector y does not have a seventh element.

R has some convenient ways to make regular vectors. Determine what each of the following does:

```r
> x = 1:10
> y = seq(0, 1, 0.01)
> z = c(rep(1, 5), rep(2, 10))
```

**Functions**

We have already used a couple of functions above. R knows a gazillion functions. A function in R is much like the functions that you have met in calculus. Namely, a function has inputs and outputs. In mathematics, we use the notation $f(x, y)$ to denote the result of apply the function $f$ to the inputs $x$ and $y$. The notation in R is quite similar. For example, if x is a vector, then `mean(x)` computes the mean of the elements of x.

```r
> y = 1:100
> mean(y)
[1] 50.5
```

There are some important extensions of the notion of function that R implements.
1. R function arguments can have optional arguments.

The trimmed mean is the mean of a set of numbers after several of the largest and smallest elements are removed. For example, the 10% trimmed mean removes 10% of the data points from each end. R computes the trimmed mean with the `mean` function using an optional argument. Try

```r
> y = c(1:9, 100)
> mean(y)
> mean(y, trim = 0.1)
```

In the above, the argument `trim=.1` says to trim the top 10% and bottom 10% of values. Experiment with various trimming values and vectors to see what you get.

2. R function arguments have names.

In the trimmed mean example, the second argument had a name `trim`. In calculus we rely on the position of the arguments to determine which input is which in functions that have several inputs. Given that R has optional arguments, some way of identifying which arguments are included is necessary. The naming of arguments serves that purpose.

The specification of which arguments are possible, optional or otherwise, can be found in the help document for the function. For `mean`, the specification is

```r
mean(x, trim = 0, na.rm = FALSE, ...)
```

It appears that there are three possible arguments: `x` which is required, `trim` which is optional (and has the default value of 0 if not included), and `na.rm` which specifies what to do with NA values. We can (and should) include optional values by using their names as above. However if we do not use names, R assumes that the arguments provided are in the order specified in the help document. So `mean(y,.1)` computes the 10% trimmed mean. R will always match up the named arguments first and then will match the unnamed arguments provided in the specified order. (This is why we did not have to name the first argument.) Try some things like:

```r
> z = 1:10
> y = z^2
> mean(y)
> mean(x = y)
> mean(y, trim = 0.1)
> mean(trim = 0.1, y)  # why does this work?
> mean(trim = 0.1, x = y)
> mean(0.1, y)  # why doesn't this work?
```

3. Many functions in R are vectorized.

If `f` is an R function that accepts numbers as input, it will often accept a vector of numbers and act “component-wise.” The result will then be a vector of the same length. Try

```r
> y = c(1:9, 100)
> mean(y)
> mean(y, trim = 0.1)
```

In the above, the argument `trim=.1` says to trim the top 10% and bottom 10% of values. Experiment with various trimming values and vectors to see what you get.
> y = c(1:10)
> log(y)

Packages

One of the reasons that R is so powerful and flexible is that users can easily add capabilities to R. So users have developed hundreds of “packages” that can easily be loaded into R for use. The package tab in the lower right window gives the packages that are available to you in the current version of Rstudio on dahl. Additional packages can be downloaded from the web. To load a package and make it available for your use, check the box corresponding to the name of the package. For now, load the packages “lattice”, “mosaic”, and “Stob”. Each of these adds some capabilities to the base version of R. For example, the ”Stob” package makes available many of the instructors favorite data sets. There is a packages pane that you can use to load packages and find information about the contents of each package. Note the contents of the “Stob” package.

Data Frames

Most of our data sets will come to us in R objects known as data frames. A data frame has rows and columns. Generally each row corresponds to one observational unit and each column corresponds to a variable. Thus each column is itself a vector. For example, the dimes data is in a data frame called ”dimes” which is in the Stob package. Notice below that the rows are numbered for our convenience – the row numbers are not part of the data frame. There are two variables in this data frame corresponding to the year the dime was minted and its mass.

> dimes

<table>
<thead>
<tr>
<th></th>
<th>Mass</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.26</td>
<td>2004</td>
</tr>
<tr>
<td>2</td>
<td>2.25</td>
<td>2004</td>
</tr>
<tr>
<td>3</td>
<td>2.25</td>
<td>1987</td>
</tr>
<tr>
<td>4</td>
<td>2.30</td>
<td>1988</td>
</tr>
<tr>
<td>5</td>
<td>2.29</td>
<td>1971</td>
</tr>
<tr>
<td>6</td>
<td>2.25</td>
<td>2007</td>
</tr>
<tr>
<td>7</td>
<td>2.27</td>
<td>2007</td>
</tr>
<tr>
<td>8</td>
<td>2.21</td>
<td>1974</td>
</tr>
<tr>
<td>9</td>
<td>2.27</td>
<td>2007</td>
</tr>
<tr>
<td>10</td>
<td>2.27</td>
<td>2004</td>
</tr>
<tr>
<td>11</td>
<td>2.27</td>
<td>1997</td>
</tr>
<tr>
<td>12</td>
<td>2.27</td>
<td>1994</td>
</tr>
<tr>
<td>13</td>
<td>2.30</td>
<td>1974</td>
</tr>
<tr>
<td>14</td>
<td>2.29</td>
<td>1996</td>
</tr>
<tr>
<td>15</td>
<td>2.27</td>
<td>1999</td>
</tr>
</tbody>
</table>
It will be very important for you to become proficient in manipulating data frames. A few functions are useful for getting information about the data frame:

```r
> str(dimes)
'data.frame': 30 obs. of 2 variables:
$ Mass: num 2.26 2.25 2.25 2.3 2.29 ...  

> dim(dimes)
[1] 30  2

> head(dimes)

                        Mass Year

It will be important to be able to access individual observational units (rows) or variables (columns) of the data frame. If df is a dataframe, df[i,j] is entry of the i\textsuperscript{th} row and j\textsuperscript{th} column. This can be generalized. To access rows:

```
> dimes[2, ]

    Mass Year
2 2.25 2004

> dimes[4:6, ]

    Mass Year
4 2.30 1988
5 2.29 1971
6 2.25 2007

To access columns:

> dimes[, 1]

[1] 2.26 2.25 2.25 2.30 2.29 2.25 2.27 2.21 2.27 2.27 2.27 2.27 2.30 2.29
[15] 2.27 2.23 2.24 2.25 2.25 2.23 2.24 2.23 2.25 2.28 2.26 2.28
[29] 2.23 2.23

> dimes[, 2]

[29] 2001 2002

An alternate way to access columns (variables) of a dataframe is the following:

> dimes$Mass

[1] 2.26 2.25 2.25 2.30 2.29 2.25 2.27 2.21 2.27 2.27 2.27 2.27 2.30 2.29
[15] 2.27 2.23 2.24 2.25 2.23 2.24 2.23 2.25 2.28 2.26 2.28
[29] 2.23 2.23

Note that each column of this dataframe is a vector and so can be used as an argument just as any vector.

> mean(dimes$Mass)

[1] 2.26

> mean(dimes[, 1])
The dataframe `counties` in the “Stob” package has census data (2000) on all the counties in the US. Investigate the population of these counties.

The dataframe `bballgames03` has data of all baseball games played in 2003. Investigate this data set. (Think about what functions you might want and see if they exist!)

### Histograms

A histogram is a graphical representation of the distribution of a quantitative variable. The input to a histogram is a vector. The data frame `sr` in the Stob package has data on all graduating Calvin College seniors of 2007. Try the following. (Note that the vector GPA in the data frame `sr` has the GPA of all graduating seniors.) The function `histogram` is in the `lattice` package.

```r
> histogram(sr$GPA)
```

If you look at the help document, `histogram` has a boatload of possible options.