```
> x <- c(4.3,4.6,4.8,5.4)
```

Let's start by putting some data into R. R stores data as scalars (a single number), vectors (a list of numbers), matrices (a table of numbers), and other ways. The function `c()` takes its arguments and puts them together into a vector; I think of it as a shortcut for "combine" or "concatenate" or something like that. The form `<-` is assignment; it means take whatever is on the right and assign it to the variable whose name is given on the left.
To summarize, this command combines 4 numbers into a vector and then assigns that to a variable named `x`. These numbers are the phosporus values for the 15 day plants.

```
> y = c(5.3,5.7,6.0,
6.3)
```

You can also use an equals sign instead of the assignment arrow (this is standard in many programing languages, but it does lead to semantically correct statements like `y=y+1`). You can also extend a command over more than one line (but some GUI front ends can make it a bit challenging to do). You can make your lines as long as you like.
These are the phosporus values for the 28 day plants.

```
> t.test(x,y)
```

You may enter this command to do a two-sample t-test. Note that by default R uses an unpooled estimate of variance (the Welch version with fractional degrees of freedom) and a two-sided alternative. You can also get a pooled estimate and upper or lower alternatives (i.e., `x` has greater mean or lesser mean).
The unpooled (Welch) version is generally the better option, but ANOVA is a generalization of the unpooled version.

```
Welch Two Sample t-test

data:  x and y
t = -3.3273, df = 5.958, p-value = 0.01602
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.8234804 -0.2765196
sample estimates:
mean of x mean of y
    4.775     5.825
```

```
> t.test(x,y,alternative="greater",var.equal=TRUE)
```

Do with x mean greater than y mean alternative and assume equal variances.

```
Two Sample t-test

data:  x and y
t = -3.3273, df = 6, p-value = 0.992
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.663206       Inf
sample estimates:
mean of x mean of y
    4.775     5.825
```

```
> pt(-3.3273,6,lower.tail=FALSE)
```

`pt()` gives you the cumulative probability (area to the left) for Student's t distribution; the `lower.tail=FALSE` option gives you the upper tail probability. The first argument is the t value, the second is the degrees of freedom.
In general in R, `pFOO(q,parms)` gives you the cumulative probability up to `q` for distribution FOO, `qFOO(p,parms)` gives you the quantile that gives you cumulative probability `p`, and `rFOO(n,parms)` gives you a random sample of size `n` from distribution FOO. Thus, we have `pt`, `pnorm`, `pf`, `pchisq`, `pbinom`, and many others as well as their `q` and `r` forms.

```
[1] 0.99207
```

```
> #
```

We now need to install and load packages. These are two separate steps. The first step (installing) gets the package onto your computer. The second step (loading) tells R that we want to use the package now in the current session. In principle you only have to install a package once, but you need to load it in each session where you need it.
The easiest way to install a standard package is to use the package installer from the menu bar. The easiest way to load is to use the package manager from the menu bar.
The manual way to install is to use the `install.packages()` function. The manual way to load is to use the `library()` function.
You saw how do do the installation in the first lab.

```
> library(cfcdae);library(Stat5303libs)
```

Loading the cfcdae and Stat5303liobs packages will automatically load several other packages, including one called "perm". We now want to do some randomization tests, also called permutation tests. The "perm" library in R does some of what we want, so loading Stat5303libs should also make the functions in perm available to us. You can use the package manager to load Stat5303libs, or you can do it with the `library()` function.

```
> permTS(x,y)
```

This function does the two-sample randomization (permutation) t-test. By default it does a two-sided alternative.
We see that the x mean is less than the y mean, and that the probability that a randomization leads to a difference of means as large or larger than 1.05 in absolute value is 5.7%. Note that this is not identical to the t-test p-value.

```
Exact Permutation Test (network algorithm)

data:  x and y
p-value = 0.05714
alternative hypothesis: true mean of x minus mean of y is not equal to 0
sample estimates:
mean of x minus mean of y
                    -1.05
```

```
> permTS(x,y,alternative="greater")
```

We may also specify different alternatives.

```
Exact Permutation Test (network algorithm)

data:  x and y
p-value = 0.9857
alternative hypothesis: true mean of x minus mean of y is greater than 0
sample estimates:
mean of x minus mean of y
                    -1.05

> #
```

OK, now we're going to enter in the data from the runstitch times from Table 2.1. There are a bunch of different ways to do it, and we'll go through a few.

```
> runstitch <- read.table("http://www.stat.umn.edu/~gary/book/fcdae.data/exmpl2.1",
header=TRUE)
```

I'm basically very lazy, so the easiest thing to do is to read data from a file. The data from the book are all on files that you can access from the web as I have done. This is the data for example 1 from chapter 2. Exercises are ex, problems are pr.

If you have downloaded the data files from the web page onto your computer, then you can do the same thing, just leave off the "http://" bit and give a complete path the file. Alternatively, you can just use the file name for any file in the current working directory (which, again, can be changed using `setwd()`.

The `read.table` function assumes that data are in columns, and every row has an equal number of data values. The header=TRUE bit means that column labels are on the first data row.

```
> runstitch
```

The data are read into a "data frame." You can usually think of a data frame as a matrix with named columns, but it is a little fancier than that.

```
    std ergo
1  4.90 3.87
2  4.50 4.54
3  4.86 4.60
4  5.57 5.27
5  4.62 5.59
6  4.65 4.61
...
```

```
> d <- runstitch$std - runstitch$ergo
```

This computes the element by element difference of the two data vectors and assigns that into a variable named `d` (for difference).

```
> d
```

Pairwise differences.

```
 [1]  1.03 -0.04  0.26  0.30 -0.97  0.04 -0.57  1.75  0.01  0.42
 0.45 -0.80  0.39  0.25  0.18  0.95 -0.18  0.71  0.42  0.43
[21] -0.48 -1.08 -0.57  1.10  0.27 -0.45  0.62  0.21 -0.21  0.82
```

> **summary(d)**

Simple summary statistics.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.0800 -0.2025  0.2550  0.1753  0.4450  1.7500
```

> **stem(d)**

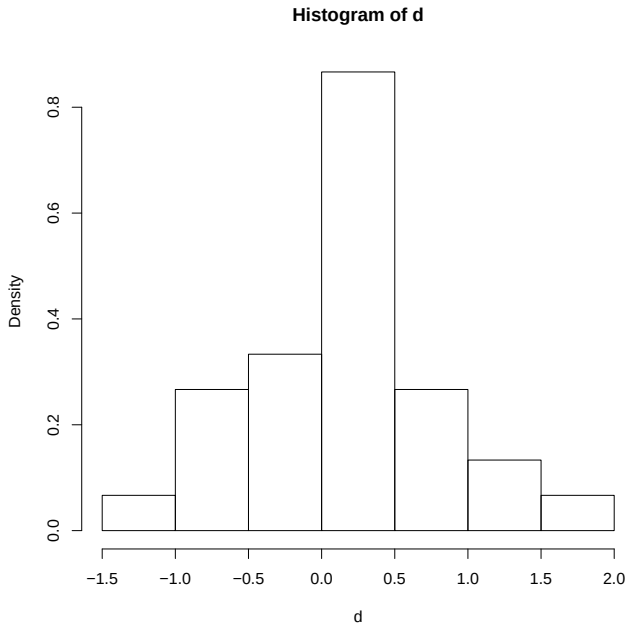
Stem and leaf plot.

```
  The decimal point is at the |

  -1 | 10
  -0 | 86655
  -0 | 220
   0 | 002233334444
   0 | 5678
   1 | 001
   1 | 8
```

> **hist(d,freq=FALSE)**

Histogram. To R's eternal shame, the default histogram is just a frequency bar chart. I want the real density. You can change the number of bins by giving a desired number of bins (e.g., hist(d,20)) or by giving a set of bin boundaries.



```
> t.test(d,alt="great")
```

The paired t-test for the full data set. This p-value is smallish, but not what we usually call significant.

```
One Sample t-test

data:  d
t = 1.49, df = 29, p-value = 0.0735
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 -0.02460234         Inf
sample estimates:
mean of x
0.1753333
```

```
> permsign.test(d,plot=TRUE)
```

This does the randomization version of the paired t, by randomly changing signs of the differences. (This was written by me, part of the Stat5303 package.) The p-values are computed by a simulation, so they will vary a bit if you repeat the command. We note that they are close to the t-test in this case.

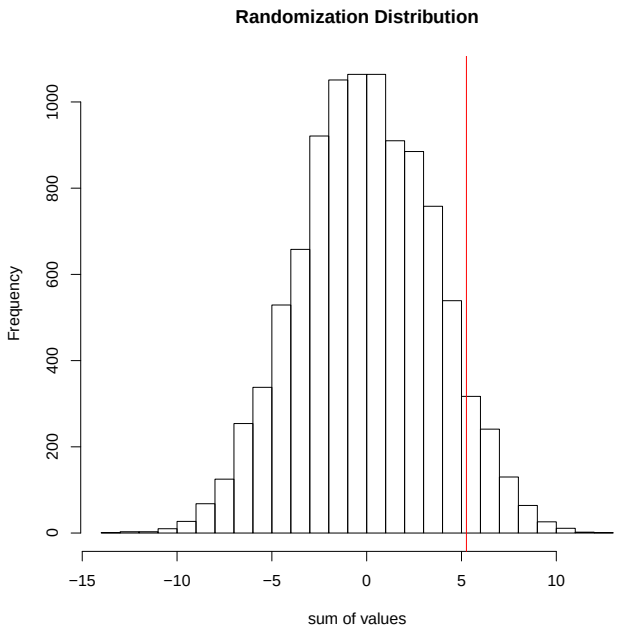
The histogram plots all of the randomization outcomes, and the red line on the plot is at the observed value.

```
$x0
[1] 5.26

$lower.p
[1] 0.93

$upper.p
[1] 0.0702

$twosided.p
[1] 0.1404
```



```
> #
```

That's it for the randomization testing lesson per se, now we will mess about some with R.

```
> runstitch["std"]
```

You can access components of the runstitch data frame by name (within square brackets). However, this gives you a new data frame that only has the requested column.

```
    std
1  4.90
2  4.50
3  4.86
```

```
> is.data.frame(runstitch["std"])
[1] TRUE
```

```
> runstitch[,2]
```

You can also access the second column of something by using ,2 within the square brackets. Note that when extracting this way, we get an ordinary vector, not a data frame.

```
 [1] 3.87 4.54 4.60 5.27 5.59 4.61 5.19 4.64 ...
```

```
> is.data.frame(runstitch[,2])
[1] FALSE
```

```
> runstitch[,"ergo"]
```

We can also select using a named column. Note that this also produces a vector rather than a data frame.

```
 [1] 3.87 4.54 4.60 5.27 5.59 ...
```

```
> runstitch$ergo
```

Using the "dollar" is another way to extract a named component. Note that this also produces a vector rather than a data frame.

```
 [1] 3.87 4.54 4.60 5.27 5.59 ...
```

```
> #install.packages("oehlert_1.02.tar.gz",repos=NULL,type="source")
```

Another way to get the book data is via an R package that you can download from the book web page. Russ Lenth put together an R package that just has all the data in it. As usual, we first need to install the package (just once). Because I have already installed this package, I have put a sharp/pound sign in front of the command. This makes anything after the sharp a comment, and R ignores it.

```
> library(oehlert)
```

Then we need to load the library (you could also do this with the package manager dialog).

```
> emp02.1
```

Then you can just give the data set name, and hey presto, Bob's your uncle, there it is. Note, Lenth used emp instead of exmpl for examples. You will also find that some of the variables are given slightly different names, and some factor levels are coded differently (usually in a more explanatory fashion). Lenth's data sets also arrive as data frames.

```
    std ergon
1  4.90  3.87
2  4.50  4.54
3  4.86  4.60
4  5.57  5.27
...
```

```
> #
```

There are many other ways to get the data in. The simplest ways are to type it all in (yuck!) using the `c()` function or to read it in from a file using the `scan()` function, which just reads all the numbers into a big vector. In either case you may need to reshape into vectors, matrix, etc. as needed.

```
> 1:10
```

The form a:b returns numbers from a up to, but not greater than b, by steps of 1. The function `seq()` can do fancier sequences.

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> df10 <- d[1:10];df10
```

We can subscript to get the first ten differences.

```
 [1]  1.03 -0.04  0.26  0.30 -0.97  0.04 -0.57
 [8]  1.75  0.01  0.42
```

```
> dl10 <- d[21:30];dl10
```

Or the last 10 differences. We will sometimes wish to work with subsets of the data.

```
 [1] -0.48 -1.08 -0.57  1.10  0.27 -0.45  0.62
 [8]  0.21 -0.21  0.82
```