Stat 202a, Statistical Computing. Prof. Rick Paik Schoenberg.

1. Administrative things.
2. R very basics.
3. Output in R.
4. Functions in R.
5. Logical operators in R.
6. Element selection and redefinition in R.
7. Vector arithmetic and order of operations in R.
8. pi.r and hw1.

1. Administrative stuff.

Adding, prerequisites, etc.
The syllabus.
The good and bad things about this course, the book, and my teaching.
Read Teetor ch. 1-4 for next class.
Read through ch. 6 of the R Cookbook for next week.
No late hw will be graded! This goes for hw 2, 3 and 4 also.

Hw1 is due Thu Oct 16 1030am by submitting through CCLE.

See the course website,
https://ccle.ucla.edu/course/view/14F-STATS202A-1 .

Stat 202a, Statistical Computing. Prof. Rick Paik Schoenberg.

2. R very basics.

2a. Downloading and installing R.

R is free and works on any major platform.
You can follow the book and use www.r-project.org, but easier is to go straight to cran.stat.ucla.edu .
Relatively easy to install. See Teetor p2. If you have problems, you can use the computers in the labs to do the first assignment. They already have R.

2b. Text editing.

It's a good idea to use a Text editor, like Alpha (or the default R text editor in the GUI version of R, or MS Word, or any other text editor), and copy and paste commands into R. To get Alpha, see http://alphatcl.sourceforge.net/wiki/pmwiki.php/Software/AlphaX . Teetor p44 tells you how to open a new editor window, but this does not always work on all platforms. See Teetor p8, Table 1-1 about moving around the command window. Up arrow and down arrow are useful.

```
  ## anything after a number sign is a comment; ; is equivalent to hitting return.
```

The common mistakes on Teetor p46-49 are a must read. p47, the problem

```
   total = 1 + 2 + 3
   + 4 + 5
```

is really common. This happens often when text editors wrap your code strangely, and then you cut and paste it. Note that if you do

```
   total = 1 + 2 + 3 +
   4 + 5
```

then the problem goes away. R is smart about that.
A related problem is this:

```
   total = 1 + 2 + 3 ## this will be the total of all my elements in
                     my dataset.
```

2c. Arithmetic.

```
2 + 3 ## arithmetic
2 * 4
2 - 5
2 / 7
sqrt(7)
7^2 ## exponents
7^(1/2)
7^1/2 ## without parentheses,
## exponents are done before multiplication and division
x = 2 ## assignment
x
x <- 3
x
x == 3
if(x == 3) y = 4
y
(x == 3) * 3.4
```

Most people use x = 3 these days, despite the ambiguity of "=" also being a logical operator.

```
pi
e
exp(1)
exp(2)
```

2d. Vectors.

```
  x = c(1,4,8) ## a vector. c means "combine".
  x
  y = c(1,4,7)
  x == y
  x = 1:50
  x
  print(x)
  Print(x) ## R cares about capitalization.
  mean(x)
  q()
```

2e. Getting help and loading packages.

help() and example() are very useful.

```
  help(exp)
  help(mean)
  help(sqrt)
  help(adf.test)
```

help(adf.test) said "no documentation for 'adf.test'".

Teetor says "The problem is that the function's package is not currently loaded."

```
  library() ## list of downloaded packages
  install.packages("spatial") ## only need to do this once (per computer)
  library(spatial) ## need this every R session when you need spatial functions
  help.search("adf.test")
  library(tseries)
  help(adf.test)
```

3. Output in R.

```
x = c("fee", "fie", "foe", "fum")
print(x)
print("The zero occurs at", 2*pi, "radians.")
```

yields an error: unimplemented type. The book solves this using repeated calls to print().
cat() is useful. Use \n for a new line.

```
cat("The zero occurs at",2*pi,"radians.\n The end.")
```

print() works with many different objects, but cat() doesn't like lists. See p25.
We will discuss this more later.

On the top of p24, Teetor says

```
print(matrix(c(1,2,3,4), 2, 2))
```

This makes a 2 by 2 matrix with the elements of the vector [1,2,3,4] read in column by column. To read them in by row, do

```
x = matrix(c(1,2,3,4), 2, 2, byrow=T)
print(x)
```

4. Functions in R.

The top of p27 gives a simple example of a function.

```
  f <- function(n,p) sqrt(p*(1-p)/n)
  f(4, .5)
```

You can also make functions have default parameter values.

```
  f = function(n,p=.5) sqrt(p*(1-p)/n)
  f(4,.5)
  f(4)
  f(4,p=.5)
```

All 3 of the above are equivalent.

All functions in R take parentheses, other than arithmetic functions like + and *, and a few other special ones. p30 lists some useful existing functions like mean, median, etc.

```
  mean(x)
  median(x)
  sd(x)
```

Note that sd goes column by column, but by default, mean and
median act on all elements together as one vector. We will see how to take
column or row means individually later, using apply().

4. Functions in R, continued.

Teetor notes on pp. 31-32 that mean(dframe) and sd(dframe) go column by column, which is usually what you want.
var() gives the covariances between columns. Same with cov().

```
x = matrix(c(1,2,3,1,4,7,1,5,1),ncol=3) ## by default, it inputs them column by column.
cov(x)
var(x)
```

seq(), rep() and : are discussed on p33. All are extremely useful.

```
10:50
x = seq(from=1,to=50,by=2)
x
length(x)
x = seq(from=1,to=50,length=15) ## to be sure to hit 50 exactly.
x
length(x)
rep(1,5)
rep(c(1,3),5)
rep(c(1,3,5),c(1,2,3))
rep(c(1,3,5),c(1,0,3))
rep(c(1,3,5),times = c(1,0,3))
```

4. Functions in R, continued.

Functions return their LAST expression, or you can specify return(x). See p42.
Variables in a function are local. x in a function is different from x outside the function. Use <<- to change or assign a variable globally.

```
x = 3
f2 = function(n){
x = 4
n*x
}

f2(5)
x

f2 = function(n){
x <<- 4
n*x
}

f2(5)
x

ca = function(b){x <<- 3; x*b}
x = 2
ca(4)
x
```

5. Logical operators in R.

a == pi evaluates whether a and pi are equal. See Teetor p34.
If they're vectors, the result is a vector of trues and falses.
!= means not equal.
<= means less than or equal to, but <- means assigment.

```
  x = 1:5
  x = 3
  x
  x = 1:5
  x == 3
  x[x==3]
  x[x!=3]
```

You can also take out one or more element of x, using -.

```
   x = seq(1,52,by=10)
   x
   x[-2]
   x[-c(2,5)]
   y = x[-c(1:6)[x > 40]]

  x & 7 ## if an element is 0, it is False, otherwise it is true.
  x[3] = 0
  x & 7
  x[3] = -1.4
  x & 7
```

6. Element selection and redefinition in R.

any() and all() are described on p35 of Teetor. There's also which().

```
  x = c(3,1,4,3,5)
  any(x == 3)
  all(x == 3)
  which(x==3)
```

Note that you can select not just one element, like fib[5] as in Teetor's example on p36, but also a collection, like fib[c(1,2,4,8)]. This is extremely useful. You can say
fib[c(1,2,4,8)] = rep(0,4) to change these values to 0.

```
  fib = c(0,1,1,2,3,5,8,13,21,34)
  fib
  fib[c(1,2,4,8)] = rep(0,4)
  fib
```

To change NA's to -1's you might say

```
  y = sqrt(fib-1)
  y[is.na(y)] = rep(-1,sum(is.na(y)))
```

%in% is described on p41.

```
  c(1:3) %in% c(14:24)
  c(1:3) %in% c(14:24,2)
```

6. Element selection and redefinition in R, continued.

```
  x = c(-1,3,5,8,-2)
  y = sqrt(x)
  is.na(y)
  y[is.na(y)] = rep(0,length(is.na(y)))
## note the error here. length(is.na(y)) is the same as the length of y.
  y[is.na(y)] = rep(0,sum(is.na(y)))
```

Or, you can use the minus sign, as on Teetor p36.

```
  x = c(-1,3,5,8,-2)
  y = sqrt(x)
  c(1:5)[is.na(y)]  ## or which(is.na(y))
  z = y[-c(1:5)[is.na(y)]]  ## or z = y[-which(is.na(y))]
     ## or z = y[which(!is.na(y))]
```

Look at c(1:5)[is.na(y)] again. If we just did

```
  1:5[is.na(y)] ## this does the wrong thing and gives you a warning.
```

Even easier, you can just use the ! sign.

```
  x = c(-1,3,5,8,-2)
  y = sqrt(x)
  z = y[!is.na(y)]
```

7. Vector arithmetic and order of operations in R.

Arithmetic on vectors goes element by element. See Teetor p38.

```
  w = (1:5)*10
  w
  w + 2
  (w+2) * 10
```

p40 gives the order of operations in R.
You might wonder what unary minus and plus are, and why they take precedence over multiplication or division. Unary means they just act on one element. For instance, the minus in the number -3 just acts on the 3.

```
  3 * - 4
  3 * + 4
```

The most important example is the one he shows on p41, 0:n-1, when n = 10.
This does 0:n first, so it creates the vector from 0 to 10, and then subtracts 1 from each element, so it's from -1 to 9. This is a VERY common mistake.

```
  n = 10
  0:n
  0:n-1
  0:(n-1)
  (0:(n-1)) ## it doesn't hurt to put parentheses around : expressions.
```

8. pi.r and other stuff for hw1 but not in Teetor ch 1-6.

As arguments to plot(), pch is useful to change the plotting symbol, and col for plotting color.
type="n" is to set up the plot but not plot the points.
points(x,y) adds the points to the current plot.
cex is for character size.
lines(x,y) connects the dots and adds them to the current plot.
lty adjusts the line type.

```
x = 1:5; y = c(3,2,4,3,4)
plot(x,y)
plot(x,y,type="n")
points(x,y,pch=".") ## tiny dots
points(x,y,pch=2) ## triangles
plot(x,y,type="n"); points(x,y,pch=3) ## plus signs
plot(x,y,type="n"); points(x,y,pch=3, col="blue")
plot(x,y,type="n"); points(x,y,pch=x, col="blue")
points(x,y,pch=as.character(x))
plot(x,y,pch=x, col="blue",cex=2); points(x,y,pch=as.character(x),cex=.7)
lines(x,y,col="red",lty=3)
```

runif(n) generates n pseudo-random uniform variables on [0,1].

```
x = runif(10000)
y = runif(10000)*20-7 ## 10000 random uniforms on [-7,13].
plot(x,y)
plot(x,y,pch=".")
quantile(x,0.9); quantile(y,0.9)
```

8. pi.r and other stuff for hw1 but not in Teetor ch 1-6, continued.

sort() sorts a vector, by default from smallest to largest.

```
  z = sort(y)
  z[1:5]
  z[9000]
  z = sort(y,decreasing=T) ## to sort from biggest to smallest.
  z[1:5]
```