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

1. Logical operators in R.
2. Element selection and redefinition in R.
3. Vector arithmetic and order of operations in R.
4. pi.r and hw1.
5. Plotting the sample mean.
6. Modes and lists.
7. Finding the statistical mode in R.
8. Working directories and libraries.

1. 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
```

1. Logical operators in R, continued.

```
help("&")
```

p48, &, |, && and || are for logical arguments.
"& and && indicate logical AND and | and || indicate logical OR.
The shorter form performs elementwise comparisons in much the same way as arithmetic operators. The longer form evaluates left to right examining only the first element of each vector."

Basically, in my personal experience, you use && within if(), and you use & to generate a vector of Trues and Falses.

```
x = c(1:10); y = x^2
((x<5) & (y>7))
x = 3
if(x > 2) print("greater.")
```

| means "or". Similar issues as with &.

2. 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)
```

2. 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)]
```

3. 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.
```

4. 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)
```

4. 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]
```

5. Plotting the sample mean.

cumsum() outputs the cumulative sum from i = 1 to k of a vector, as k goes from 1 to n.
Suppose we want to plot the sample mean of 200,000 iid N(0.12, 1)s.

```
n = 200000
x = rnorm(n, mean=0.12, sd = 10)
y = cumsum(x)/(1:n)
plot(y)
## this is useful to see if the sample mean has converged.
## problems: a) the line is so thick you can't see what it's converged to.
##          b) the y-axis is too broad to see what it's converged to.
##          c) the x and y labels.
plot(c(1,n), c(-0.4, 0.4), type="n",
  xlab="k", ylab="")
mtext(s=2,l=2,cex=0.7, expression(paste(frac(1,k), "  ",sum(x_i,i==1, k))))
points(1:n, y, pch=".") ## or lines(1:n,y)
abline(h=mean(x),lty=2)

mean(x)
## By the central limit theorem,
## the standard error is sigma/sqrt(n) = 1/sqrt(200000) ~ 0.002236,
## and a 95% range for mean(x) is 0.12 +/- 1.96/sqrt(200000) ~ (0.1156, 0.1244)
se2 = 1.96*10/sqrt(1:n)
lines(1:n,y+se2,lty=2,col="blue")
lines(1:n,y-se2,lty=2,col="blue")
```

6. Modes and Lists

Two common problems discussed by Teetor p48 are aList[i] and aList[[i]], and & and &&.

```
aList = list(w = rep(0,4), x = 1:10, y = rep(3,12), z = 1:5)
aList[[2]] ## the second item in the list
aList[2] ## a list containing the second item, and usually not what you want.
x = aList[[2]]
y = aList[2]
x+1
y+1
mode(x)
mode(y)

x = 3; mode(x) ## "numeric".
x = "a"; mode(x) ## "character".
```

7. Finding the statistical mode, in R.

mode(x) gives you the type of variable x is, not a vector's most commonly appearing element. How do find the mode, in the statistical sense?

We can find the mode using table(). table(x) gives a list of the sorted elements in x, along with their counts.

```
x = c(17,4.3,2.1,4.3,1)
table(x)
y = which.max(table(x)) ## Outputs the index in sorted list of x, not the mode.
y = table(x)
names(y)
z = as.numeric(names(y)) ## equivalent to z = sort(unique(x))
z[which.max(y)]

mode2 = function(x){ ## finds the mode, but if there's a tie, defaults to the smallest mode.
y = table(x)
z = as.numeric(names(y))
z[which.max(y)]
}

x = c(17,17,4,4,2)
mode2(x)
```

```
  mode3 = function(x){ ## finds the mode(s)
  y = table(x)
  z = as.numeric(names(y))
  y1 = as.numeric(y)
  w = (y1 == max(y1))
  z[w]
  }

## equivalently, we could replace w = (y1 == max(y1)) with
## w = which(y1 == max(y1))

  mode3(x)
  x = c(rep(1,7), rep(10,7), rep(-1,8))
  mode3(x)
  x1 = x[x != -1]
  mode3(x1)
```

8. Working directories and libraries.

Setting the working directory is really important to read files or write to files and know where they go. See Teetor p51.

p54 search() lists all packages currently loaded [not just installed but loaded in current session].

```
search()
library(MASS)
search()
detach(package:MASS)
search()
```

Loading in datasets is just like loading packages, see Teetor p53.

```
head(pressure) ## or just pressure or pressure[1,]
data() ## lists all the preloaded datasets.
instead of data(Cars93, package="MASS") you could just do
library(MASS)
data(Cars93)
```

p58, library() lists all installed (but not necessarily loaded) packages.
install.packages() is useful to install a new one.
p63, source() is sometimes useful though it can have problems,
especially if your comments go over lines, as in the example with

```
## this will be the total of all
```

the elements in my dataset.