Dataframes: merge

```
tb1 = data.frame(indiv_id = 1:4, snp1 = c(1,1,0,1), snp2 = c(1,1,0,0))

tb2 = data.frame(indiv_id = c(1,3,4,6), cov1 = c(1.14,4.50,0.80,1.39),
                                          cov2 = c(74.6,79.4,48.2,68.1))


tb1
  indiv_id snp1 snp2
        1    1    1
        2    1    1
        3    0    0
        4    1    0


tb2
  indiv_id cov1 cov2
        1 1.14 74.6
        3 4.50 79.4
        4 0.80 48.2
        6 1.39 68.1

merge(tb1, tb2, by="indiv_id", all=TRUE)

  indiv_id SNP1 SNP2 cov1 cov2
        1    1    1 1.14 74.6
        2    1    1   NA   NA
        3    0    0 4.50 79.4
        4    1    0 0.80 48.2
        6   NA   NA 1.39 68.1
```

Dataframes: merge

```
tb1
  indiv_id snp1 snp2
        1    1    1
        2    1    1
        3    0    0
        4    1    0

tb2
  indiv_id cov1 cov2
        1 1.14 74.6
        3 4.50 79.4
        4 0.80 48.2
        6 1.39 68.1

merge(tb1, tb2, by="indiv_id", all.x=T)

  indiv_id snp1 snp2 cov1 cov2
1        1    1    1 1.14 74.6
2        2    1    1   NA   NA
3        3    0    0 4.50 79.4
4        4    1    0 0.80 48.2
```

Dataframes: merge

```
tb1
  indiv_id snp1 snp2
        1    1    1
        2    1    1
        3    0    0
        4    1    0

tb2
  indiv_id cov1 cov2
        1 1.14 74.6
        3 4.50 79.4
        4 0.80 48.2
        6 1.39 68.1

merge(tb1, tb2, by="indiv_id", all.y=T)

  indiv_id snp1 snp2 cov1 cov2
1        1    1    1 1.14 74.6
2        3    0    0 4.50 79.4
3        4    1    0 0.80 48.2
4        6   NA   NA 1.39 68.1
```

Dataframes: merge

```
names(tb2) = c("id","cov1","cov2")

tb1
  indiv_id snp1 snp2
1        1    1    1
2        2    1    1
3        3    0    0
4        4    1    0

tb2
  id cov1 cov2
1  1 1.14 74.6
2  3 4.50 79.4
3  4 0.80 48.2
4  6 1.39 68.1

merge(tb1,tb2,by.x="indiv_id",by.y="id",all=TRUE)
  indiv_id snp1 snp2 cov1 cov2
1        1    1    1 1.14 74.6
2        2    1    1   NA   NA
3        3    0    0 4.50 79.4
4        4    1    0 0.80 48.2
5        6   NA   NA 1.39 68.1
```

Dataframes: split

The split function lets us break up a data frame based on a grouping variable. We'll look at this in greater detail in the aggregate section but for now let's focus on how to do this.

Let's say we want to split up mtcars based on the number of cylinders which take on the values 4,6,8. We use the split command to do this and what it gives back to us is a list with each element containing a part of the data frame corresponding to each cylinder group.

We could use the subset command or bracket notation to pull out the information

```
eight.cyl = mtcars[mtcars$cyl == 8,]

six.cyl = mtcars[mtcars$cyl == 6, ]

four.cyl = mtcars[mtcars$cyl == 4, ]
```

Dataframes: split

```
(hold = split(mtcars, mtcars$cyl) )
hold
$`4`
                mpg cyl  disp hp drat    wt  qsec vs am gear carb
Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1
Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1

$`6`
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4

$`8`
                    mpg cyl  disp  hp drat   wt  qsec vs am gear carb
Hornet Sportabout  18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
Duster 360         14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
Merc 450SE         16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
Merc 450SL         17.3   8 275.8 180 3.07 3.73 17.60  0  0    3    3
Merc 450SLC        15.2   8 275.8 180 3.07 3.78 18.00  0  0    3    3
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.25 17.98  0  0    3    4
```

Dataframes: split

```
hold = split(mtcars, mtcars$cyl)

sapply(hold,nrow)
 4  6  8
11  7 14
```

Why is this useful ? Well we might want to focus in on only the cars occupying a certain cylinder group while ignoring the rest. So if we wanted only the 8 cylinder cars:

```
eight.cyl = hold$`8`

-OR-

eight.cyl = hold[[3]]
```

Dataframes: split

We could also use this approach to do some summary reporting. This might seem advanced at this point but its good for you too see this kind of approach as it is common in R. This example gives the mean MPG for each cylinder group:

```
hold = split(mtcars,mtcars$cyl)
sapply(hold, function(x) mean(x$mpg))

   4        6        8
26.66364 19.74286 15.10000
```

We can use a more complicated function of our own design. Like take the mean per group of only automatic transmission cars.

```
my.func <- function(x) {
    hold = x[x$am == 0,]
    retvec = c(mean=mean(hold$mpg),sd=sd(hold$mpg))
    return(retvec)
}

sapply(hold, my.func)
             4         6         8
mean 22.900000 19.125000 15.050000
sd    1.452584  1.631717  2.774396
```

Dataframes - order/sort

Let’s take a look at what the order command does. It returns the record/row numbers of the data frame from lowest MPG to highest. So record #15 must be the lowest MPG automobile in the set. And record #20 must have the highest MPG

```
order(mtcars$mpg)
 [1] 15 16 24  7 17 31 14 23 22 29 12 13 11  6  5 10 25 30  1  2  4 32 21  3  9
[26]  8 27 26 19 28 18 20


mtcars[15,]
                    mpg cyl disp  hp drat   wt  qsec vs am gear carb
Cadillac Fleetwood 10.4   8  472 205 2.93 5.25 17.98  0  0    3    4

mtcars[20,]
                mpg cyl disp hp drat    wt qsec vs am gear carb
Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.9  1  1    4    1
```

Dataframes - order/sort

Ordering and sorting data frames is an important technique

```
# sort by mpg (ascending)
newdata = mtcars[order(mtcars$mpg),]

                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4

newdata = mtcars[rev(order(mtcars$mpg)),]

                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2

newdata = mtcars[order(-mtcars$mpg),]

head(newdata)
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

Dataframes - order/sort

You can sort by multiple columns or "keys"

```
newdata = mtcars[order(mtcars$cyl),]

head(newdata)
                mpg cyl  disp hp drat    wt  qsec vs am gear carb
Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1
Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1

newdata = mtcars[order(mtcars$cyl,mtcars$mpg),]

head(newdata)
               mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Volvo 142E    21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Datsun 710    22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Merc 230      22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 240D     24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
```

Dataframes - order/sort

You can sort by multiple columns or "keys"

```
newdata = mtcars[order(mtcars$cyl,-mtcars$mpg),]

head(newdata)
                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
```

Dataframes - sample

The sample() function is quite useful when you want to take, well, a sample of your data. You can sample with or without replacement. The basic function works as follows:

```
# Take a random sample of something - in this case a vector of numbers from 1 to 20
my_vec = 1:20

sample(my_vec,10,replace=TRUE)    # Repetition is possible
 [1]  3 20 16 14 16 10 18  7  7  6

sample(my_vec, 10, replace=TRUE)  # Different results each time
 [1]  5  1  2  2 19  8 20 11  3 19

sample(my_vec, 10, replace=FALSE) # Don’t replace to insure unique numbers
 [1]  2  8  9  6 17 18  3  5 14 15

sample(1:20, 10, replace=FALSE)   # Short cut
 [1] 13  6  4 14  3 19 16 17 20 12
```

To sample from a data frame you will need to know the total number of records/observations. Use the **nrow()** function to get this. Then decide how many records you want to sample. You probably want only unique records in your sample so then set **replace=FALSE**

Dataframes - sample

Use the **sample( )** function to take a random sample of size n from a dataset.

```
# Take a random sample of size 10 from dataset mtcars
# Sample without replacement

my_records = sample(1:nrow(mtcars), 10, replace = FALSE)

my_records
 [1] 21  6  9 30 29 28  3 11 12  1

sample_of_ten = mtcars[my_records,]
sample_of_ten
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

Programming Structures - Intro

Goals for this session:

* Understand the for-loop structure and how to use it to:

* "Walk" through a vector while accumulating a sum, computing a product, or some other operation.

* "Walk" though a matrix by row, (or column), while accumulating a sum, computing a product or some other arithmetic operation.

* "Walk" through a data frame by row to compute something. Also process the results of a previous "split" operation.

* Understand the if statement and how to branch based on the value of a vector or matrix element.

* Also use the if statement in conjunction with the for loop to do some processing.

Programming Structures - Intro

Some suggestions for the upcoming weeks:

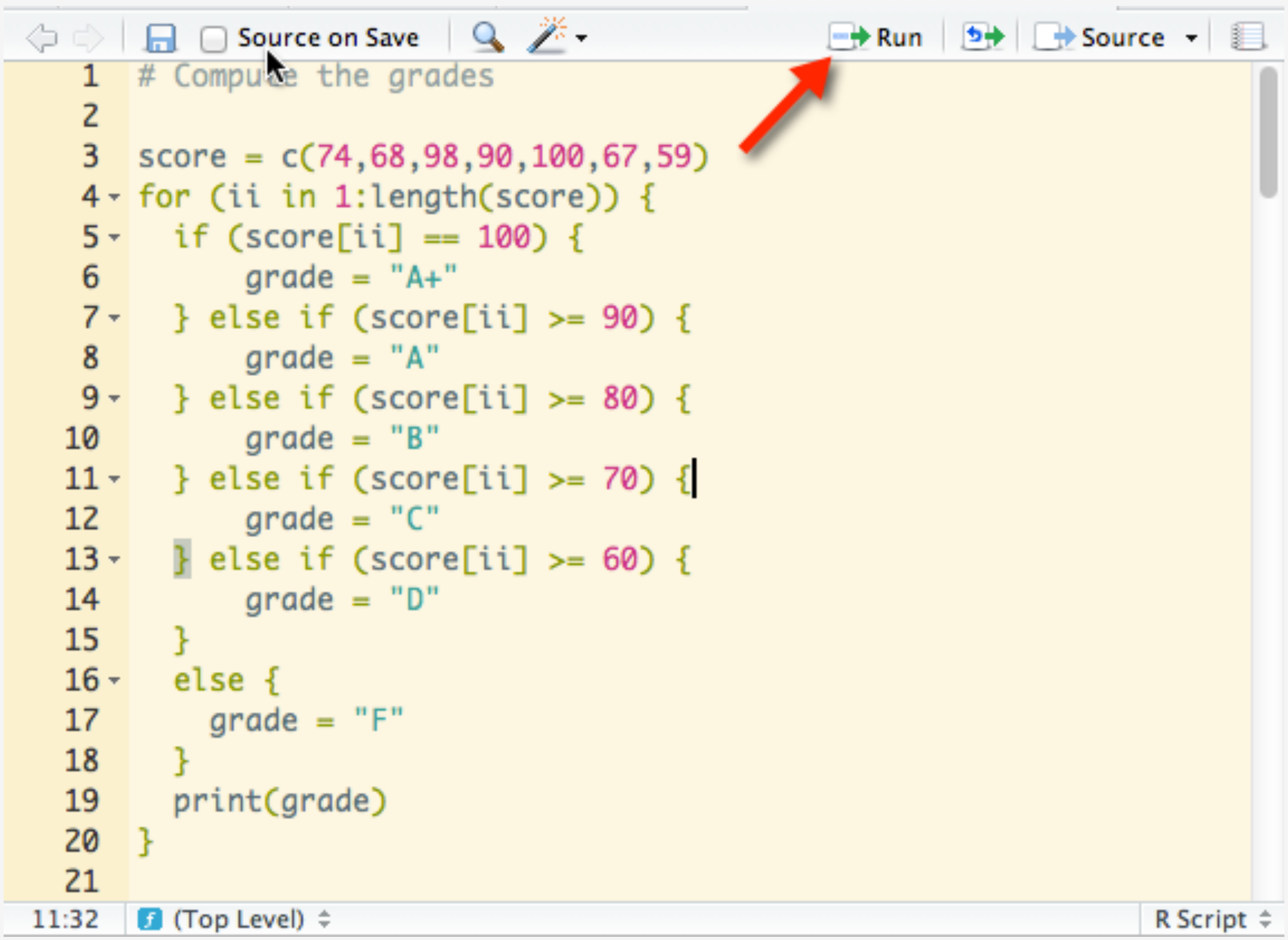
1) Put the statements in the Editor window of RStudio and perfect them there. You can highlight sections of code and hit the "run" button.

2) You will most definitely make mistakes when writing loops. It is guaranteed. Better to get familiar with the most common mistakes ahead of time.

3) Work through the labs. Use control statements and loops

4) The next assignment will assume facility with these structures.

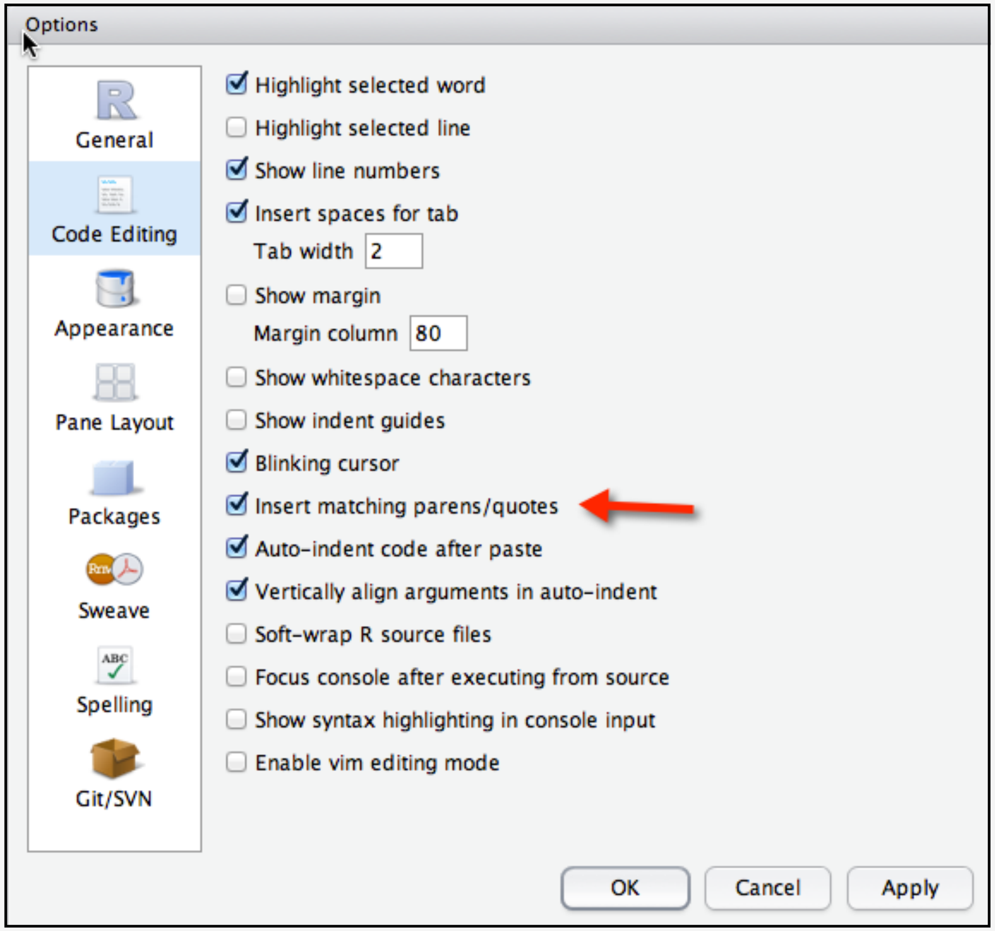
Programming Structures - Intro



```
# Compute the grades

score = c(74,68,98,90,100,67,59)
for (ii in 1:length(score)) {
  if (score[ii] == 100) {
      grade = "A+"
  } else if (score[ii] >= 90) {
      grade = "A"
  } else if (score[ii] >= 80) {
      grade = "B"
  } else if (score[ii] >= 70) {
      grade = "C"
  } else if (score[ii] >= 60) {
      grade = "D"
  }
  else {
    grade = "F"
  }
  print(grade)
}

```

Programming Structures - Intro

Go to Preferences -> Code Editing to turn on "insert matching parens/braces"



☑ Highlight selected word
□ Highlight selected line
☑ Show line numbers
☑ Insert spaces for tab
Tab width 2
□ Show margin
Margin column 80
□ Show whitespace characters
□ Show indent guides
☑ Blinking cursor
☑ Insert matching parens/quotes
☑ Auto-indent code after paste
☑ Vertically align arguments in auto-indent
□ Soft-wrap R source files
□ Focus console after executing from source
□ Show syntax highlighting in console input
□ Enable vim editing mode

Programming Structures - for

This is a looping construct that let’s you do some things for a specific number of times. “name” is some index variable that takes on values returned by “expr_1”, which is almost always some type of sequence. It could represent the length of a vector or a row of a matrix.

```
for (name in expr_1) {
   expr_2
}

x = 1:3
for (ii in 1:3) {
  print(ii)
}

[1] 1
[1] 2
[1] 3
```

Programming Structures - for with vectors

Better to generalize this - use the length function so it will work with any vector

```
x = 1:3
for (ii in 1:length(x)) {
  print(ii)
}

[1] 1
[1] 2
[1] 3
```

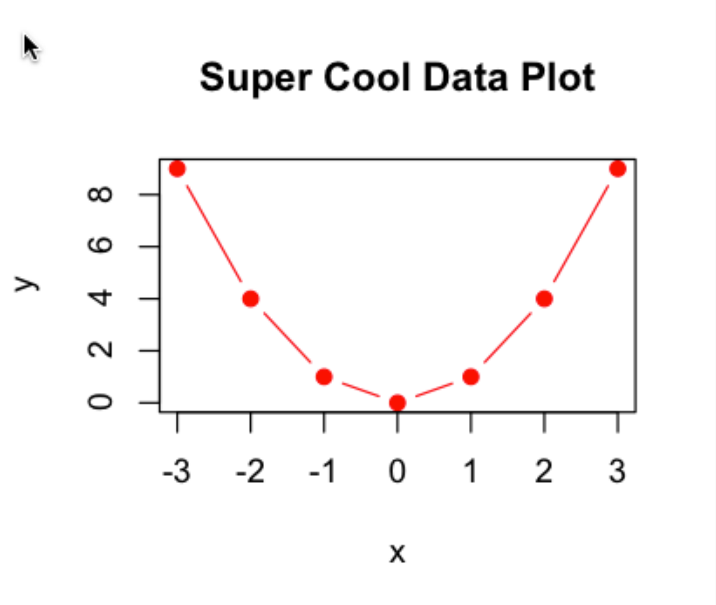
We could go backwards also:

```
x = 1:3
for (ii in length(x):1) {    # We start with the last element number
   print(ii)
}
[1] 3
[1] 2
[1] 1
```

Programming Structures - for with vectors

Let' look at the situation where we have a bunch of x values that we want to provide as input to some function. That is - We want to generate y values for plotting x vs y. Here let's plug the x values into the function x^2. (The resulting plot will be a parabola).

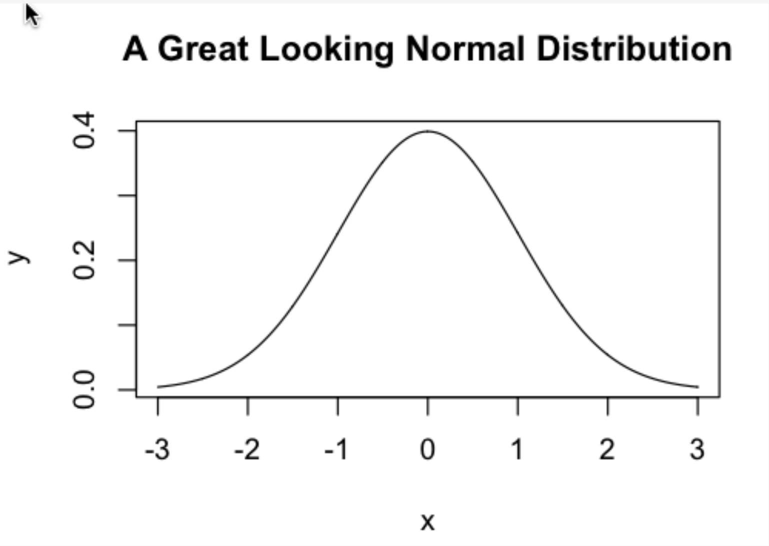
```
y = vector()  # A blank vector
x = -3:3
for (ii in -3:length(x)) {
  y[ii] = x[ii]^2
}

x
[1] -3 -2 -1  0  1  2  3

y
[1]  9  4  1  0  1  4  9
```



```
plot(x,y,main="Super Cool Data Plot",type="b",pch=19,col="red")
```

Programming Structures - for with vectors

```
y = vector()
x = seq(-3,3,by=0.005)      # seq let's us specify and increment
for (ii in -3:length(x)) {
  y[ii] = dnorm(x[ii])
}

length(x)
[1] 1201

plot(x,y,main="A Great Looking Normal Distribution",type="l")
```



Programming Structures - for with vectors

Now - we would usually use vector arithmetic to do this but we can easily add things up and take an average.

```
x = rnorm(1000,20,4)  # 1,000 random elements from a N(20,4)

mysum = 0
for (ii in 1:length(x)) {
  mysum = mysum + x[ii]
}
avg = mysum / length(x)
cat("The average of this vector is:",avg,"\n")

[1] "The average of this vector is: 20.1320691898645"
```

We could clean up the output a little bit

```
cat("The average of this vector is:",round(avg,2),"\n")
[1] "The average of this vector is: 20.13"
```

Programming Structures - for with vectors

Given a vector find the smallest value without using the "min" function:

```
set.seed(188)
x = rnorm(1000)  # 1,000 random elements from a N(20,4)

mymin = x[1] # Set the min to the first element of x. Unless we are
             # very lucky then this will change as we walk through
             # the vector

for (ii in 1:length(x)) {
  if (x[ii] < mymin) {
     mymin = x[ii]
  }
}

mymin
[1] -3.422185

min(mymin)       # The internal R function matches what we got
[1] -3.422185
```

Programming Structures - for with dataframes

We can loop through data frames also. Let's see if we can compute the mean of the MPG for all cars. Note that we use the nrow function to get the number of rows to loop over.

```
mpgsum = 0
for (ii in 1:nrow(mtcars)) {
  mpgsum = mpgsum + mtcars[ii,"mpg"]
}

mpgmean = mpgsum/nrow(mtcars)   # Divide the sum by the # of records

cat("Mean MPG for all cars is:",mpgmean,"\n")

Mean MPG for all cars is: 20.09062

all.equal(mpgmean,mean(mtcars$mpg))
[1] TRUE
```

Programming Structures - for with dataframes

Remember the split command ? We can work with the output of that also. Relative to mtcars we let's split up the data frame by cylinder number, which is (4,6, or 8).

```
mysplits = split(mtcars, mtcars$cyl)

str(mysplits, max.level=1)
List of 3
 $ 4:'data.frame':    11 obs. of  11 variables:
 $ 6:'data.frame':    7 obs. of  11 variables:
 $ 8:'data.frame':    14 obs. of  11 variables:
```

We get back a list that contains 3 elements each of which has a data frame corresponding to the number of cylinders. If we wanted to we could summarize each of these data frame elements using a for loop

Programming Structures - for with dataframes

```
mysplits
$`4`
               mpg cyl  disp hp drat    wt  qsec vs am gear carb
Merc 240D     24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2
Merc 230      22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2
Toyota Corona 21.5   4 120.1 97 3.70 2.465 20.01  1  0    3    1
..
..

$`6`
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
..
..

$`8`
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
..
..
```

Programming Structures - for with dataframes

```
mysplit = split(mtcars,mtcars$cyl)

for (ii in 1:length(mysplit)) {
   print(nrow(mysplit[[ii]]))
}

[1] 11
[1] 7
[1] 14

# This is equivalent to

sapply(mysplit, nrow)
 4  6  8
11  7 14
```

Programming Structures - for with dataframes

```
mysplit = split(mtcars,mtcars$cyl)

for (ii in 1:length(mysplit)) {
   splitname = names(mysplit[ii])
   cat("mean for",splitname,"cylinders is",mean(mysplit[[ii]]$mpg),"\n")
}
mean for 4 cylinders is 26.66364
mean for 6 cylinders is 19.74286
mean for 8 cylinders is 15.1

# This is basically equivalent to

sapply(mysplit, function(x) mean(x$mpg))
       4        6        8
26.66364 19.74286 15.10000
```

Programming Structures - for with dataframes

What about looping over each split and pulling out only those cars with an manual transmission ? (am == 1)

```
data(mtcars)

mysplit = split(mtcars,mtcars$cyl)

mylist = list() # Setup a blank list to contain the subset results

for (ii in 1:length(mysplit)) {
  mylist[[ii]] = subset(mysplit[[ii]], am == 1)
}

mylist

# Equivalent to:

lapply(mysplit, subset, am == 1)
```

Programming Structures - for with dataframes

What about looping over each split and sampling two records from each group ?

```
for (ii in 1:length(mysplits)) {
    recs = sample(1:nrow(mysplits[[ii]]),2,F)
    print(mysplits[[ii]][recs,])
}
             mpg cyl disp hp drat    wt  qsec vs am gear carb
Honda Civic 30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
Fiat 128    32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1

              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4

                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

Programming Structures - for with dataframes

What about looping over each split and sampling two records from each group ?

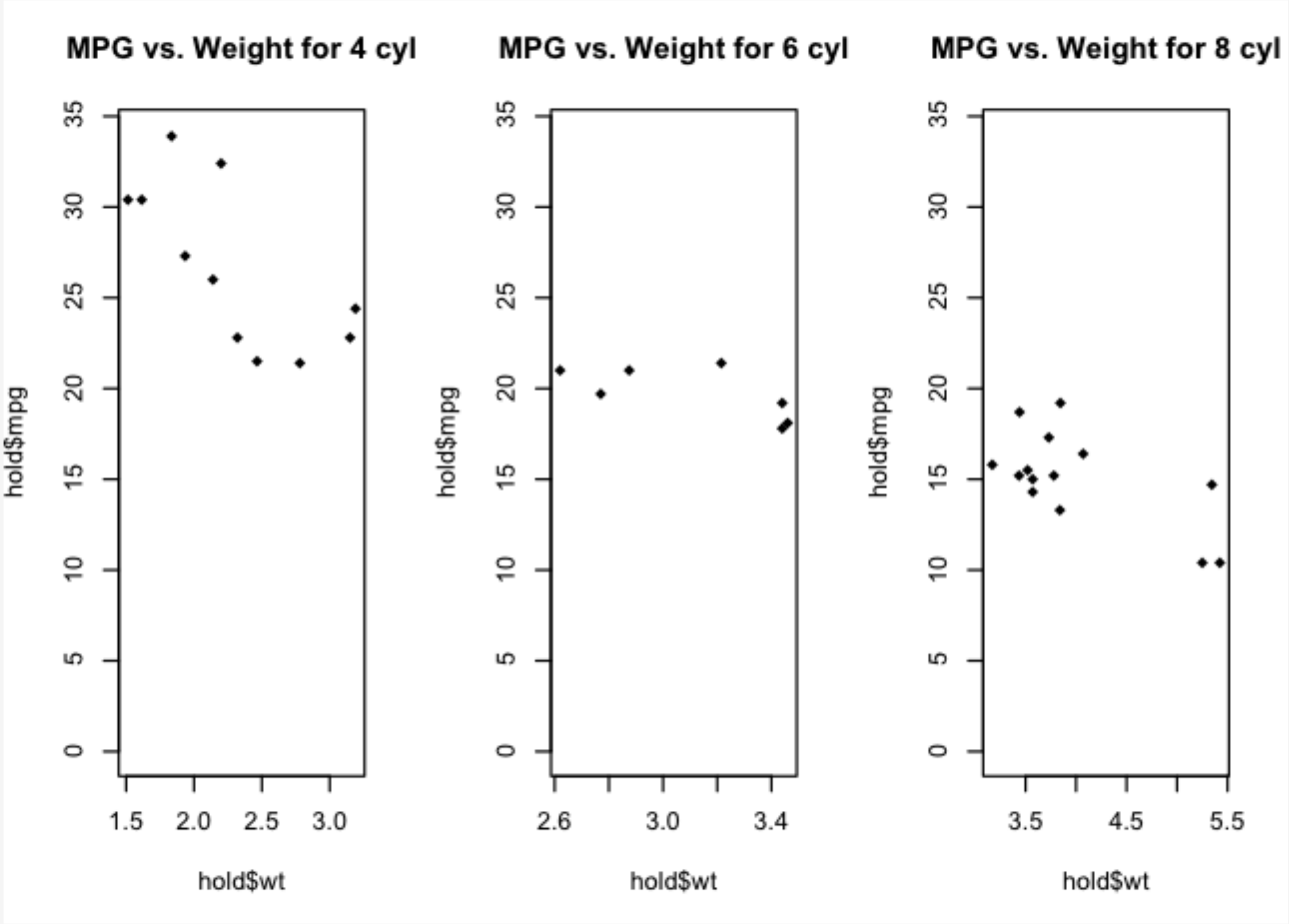
```
lapply(mysplit, function(x) {
                      recs = sample(1:nrow(x),2,F)
                      return(x[recs,])
                  })
$`4`
             mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Volvo 142E  21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2

$`6`
          mpg cyl  disp  hp drat   wt  qsec vs am gear carb
Merc 280 19.2   6 167.6 123 3.92 3.44 18.30  1  0    4    4
Valiant  18.1   6 225.0 105 2.76 3.46 20.22  1  0    3    1

$`8`
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Duster 360       14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Pontiac Firebird 19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
```

Programming Structures - for with dataframes

Let's say we want to plot MPG vs. Weight for each cylinder group. Check it out:

```
mysplits = split(mtcars, mtcars$cyl)

par(mfrow=c(1,3))    # This relates to plotting

for (ii in 1:length(mysplits)) {
  hold = mysplits[[ii]]
  plot(hold$wt, hold$mpg, pch = 18, main=paste("MPG vs. Weight for",
       names(mysplits[ii]), "cyl",sep=" "),ylim=c(0,34))
}

NOTE:
names(mysplits[1])
[1] "4"
names(mysplits[2])
[1] "6"
names(mysplits[3])
[1] "8"
```

Programming Structures - for with dataframes



Programming Structures - for with dataframes

We could write our own version of split using for loops. Try it and see:

```
hold = list()

kk = 1

for (ii in unique(mtcars$cyl)) {
     hold[[kk]] = mtcars[mtcars$cyl == ii,]
     kk = kk + 1
}

names(hold) = unique(mtcars$cyl)
```

Programming Structures - for with matrices

```
set.seed(123)

mymat = matrix(round(rnorm(6),2),3,2)

      [,1] [,2]
[1,] -0.56 0.07
[2,] -0.23 0.13
[3,]  1.56 1.72


for (ii in 1:nrow(mymat)) {
  for (jj in 1:ncol(mymat)) {
   cat("The value at row",ii,"and column",jj,"is",mymat[ii,jj],"\n")
  }
}

The value at row 1 and column 1 is -0.56
The value at row 1 and column 2 is 0.07
The value at row 2 and column 1 is -0.23
The value at row 2 and column 2 is 0.13
The value at row 3 and column 1 is 1.56
The value at row 3 and column 2 is 1.72
```

Programming Structures - for with matrices

Let's say we wanted to sum all the rows:

```
      [,1] [,2]
[1,] -0.56 0.07
[2,] -0.23 0.13
[3,]  1.56 1.72

rowtotal = 0                  # initialize a variable to hold row total
for (ii in 1:nrow(mymat)) {
  for (jj in 1:ncol(mymat)) {
    rowtotal = rowtotal + mymat[ii,jj]
  }
  print(rowtotal)
  rowtotal = 0
}

[1] -0.49
[1] -0.1
[1] 3.28

                    # same values as:
apply(mymat,1,sum)
[1] -0.49 -0.10  3.28
```

Programming Structures - if

This is an easy structure. It tests for a conditions and, based on that, executes a specific block of code.

```
if (logical_expression) {
   do something
   ...
}

if (logical_expression) {
  do something
  ..
} else {
  do something else
  ...
}
```

Programming Structures - if

```
x = 3

x
[1] 3

if (is.numeric(x)) {
   print("x is a number")
}

[1] "x is a number"

if (x != 3) {
     print("x is not equal to 3")
} else {
    print("guess what ? x is in fact equal to 3")
}
[1] "guess what ? x is in fact equal to 3"
```

Programming Structures - if

This is an easy structure. It tests for a conditions and, based on that, executes a specific block of code.

```
some.num = 3



if (some.num < 3) {        # A more involved if statement
     print("Less than 3")
} else if (some.num > 3) {
     print("Greater than 3")
} else {
     print("Must be equal to 3")
}
[1] "Must be equal to 3"
```

Programming Structures - error checking

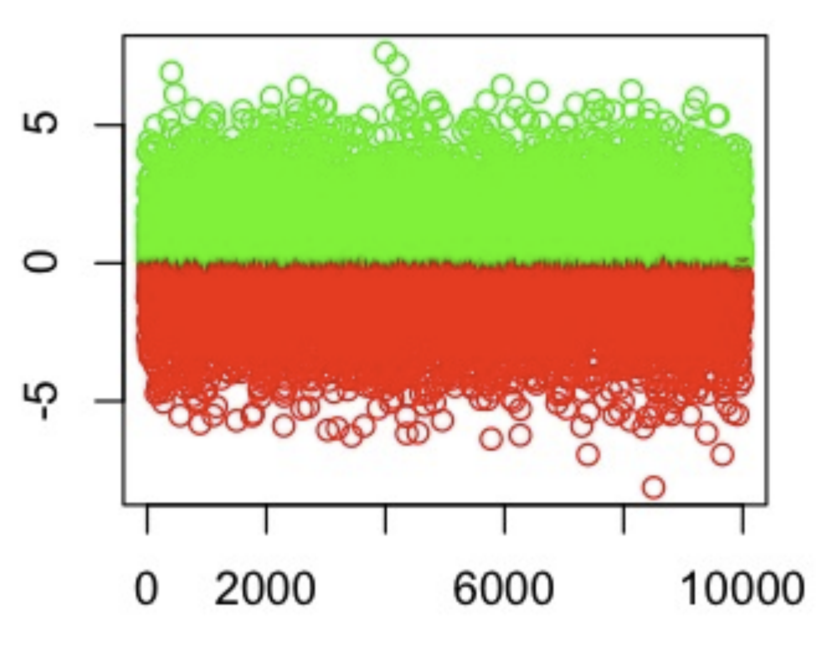
if statements show up in error checking.

```
x=4 ; y=5

if (!is.numeric(x) | !is.numeric(y)) {
   stop("I need numeric values to do this")
} else {
   if (x == y) {
       print("Equal")
   } else {
      print("Not equal")
   }
}

[1] "Not equal"
```

Programming Structures - ifelse()

The ifelse command was designed to operate on vectors. Its very fast. It has the format of **ifelse(test, yes, no)**

```
some.data = rnorm(10000,0,2)
colors = ifelse(some.data < 0,"RED","GREEN")
plot(some.data,col=colors)

# This would be the same as:

for (ii in 1:length(some.data)) {
    if (some.data[ii] < 0) {
       colors[ii] = "RED"
    } else {
       colors[ii] = "GREEN"
    }
}
```



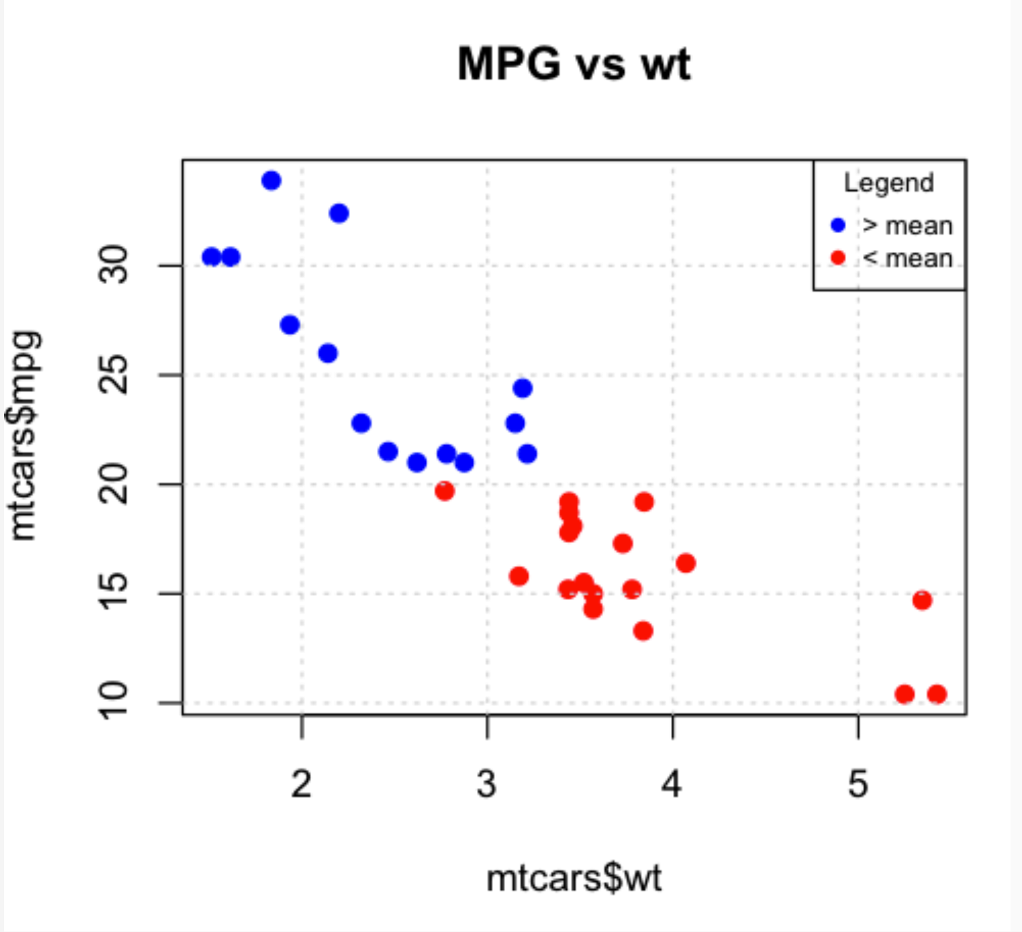
Programming Structures - ifelse()

We use ifelse when want to segregate some continuous quantity within a data frame into a factor that we can then use within other R functions.

```
mtcars$rating = ifelse(mtcars$mpg >= mean(mtcars$mpg), "blue", "red")

head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb rating
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4   blue
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4   blue
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   blue
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   blue
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    red
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    red

plot(mtcars$mpg~mtcars$wt,col=mtcars$rating,pch=19, main="MPG vs wt")

grid()

legend("topright", c("> mean","< mean"), pch=19,
        col=c("blue","red"),title="Legend",cex=0.7)
```

Programming Structures - ifelse()

We use ifelse when want to segregate some continuous quantity within a data frame into a factor that we can then use within other R functions.



Programming Structures - for and if

So if statements are usually part of some other structure - like within a for-loop:

```
score = c(74,68,98,90,100,67,59)

for (ii in 1:length(score)) {
 if (score[ii] == 100) {
   grade = "A+"
 } else if (score[ii] >= 90) {
   grade = "A"
 } else if (score[ii] >= 80) {
   grade = "B"
 } else if (score[ii] >= 70) {
   grade = "C"
 } else if (score[ii] >= 60) {
   grade = "D"
 }
 else {
  grade = "F"
 }
 print(grade)
}
[1] "C"
[1] "D"
[1] "A"
[1] "A"
[1] "A+"
[1] "D"
[1] "F"
```

Programming Structures - for and if

So if statements are usually part of some other structure - like within a for-loop:

```
set.seed(123)
x = round(runif(10,1,20))
[1]  6 16  9 18 19  2 11 18 11 10

for (ii in 1:length(x)) {
    if (x[ii] %% 2 == 0) {
        print(TRUE)
    }
    else {
        print(FALSE)
    }
}

[1] TRUE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
```

Programming Structures - for and if

We can mimic the bracket notation approach here:

```
set.seed(123)
x = round(runif(10,1,20))
 [1]  7 13 16  8 13 19  9 11 13 11

logvec = vector()            # Setup an empty vector
for (ii in 1:length(x)) {
    if (x[ii] %% 2 == 0) {
        logvec[ii] = TRUE
    }
    else {
        logvec[ii] = FALSE
    }
}
logvec

logvec
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE


x[logvec]
[1]  6 16 18  2 18 10
```

Programming Structures - for and if

One can easily “break” out of a for loop based on some condition. Normally you should clean your data before processing it but perhaps you thought you did. Let’s say that you are processing elements of a vector and if you encounter a value of NA then you want to stop the for loop.

```
my.vec = c(1,2,3,NA,5,6,7,8,9,10)

for (ii in 1:length(my.vec)) {
    if (is.na(my.vec[ii])) {
       break
    }
    cat("element is ",ii,"\n")
}

element is  1
element is  2
element is  3
```

Programming Structures - for and if

Here we want to "catch" the missing value and then "skip over it". To do this we would use the "next" statement.

```
my.vec = c(1,2,3,NA,5,6,7,8,9,10)


for (ii in 1:length(my.vec)) {
    if (is.na(my.vec[ii])) {
       next
    }
    cat("element is ",ii,"\n")
}

element is  2
element is  3
element is  5
element is  6
element is  7
element is  8
element is  9
element is  10
```

Programming Structures - for and if

Here is an example that will be useful when processing things like genetic sequences. Let's say we have a string of text we wish to "encode" by changing all vowels to something else. This isn't a tough code to break but let's see what is involved. In our code:

We'll change a to s,
e to t,
i to u,
o to v,
u to w

So a string like :

sequence = "Hello my name is Ed. Happy to meet you"

would come out like:

```
"Htllv my nsmt us td. Hsppy tv mttt yvw"
```

Programming Structures - for and if

```
sequence = "Hello my name is Ed. Happy to meet you"

seq = unlist(strsplit(sequence,""))

[1] "H" "e" "l" "l" "o" " " "m" "y" " " "n" "a" "m" "e" " " "i" "s" " " "E"
"d" [20] "." " " "H" "a" "p" "p" "y" " " "t" "o" " " "m" "e" "e" "t" " " "y"
"o" "u"

sequence = "Hello my name is Ed. Happy to meet you"
seq = unlist(strsplit(sequence,""))
for (ii in 1:length(seq)) {

  # Write code to inspect each element of seq to determine if it is
  # a candidate for changing.

}



"Htllv my nsmt us Ed. Hsppy tv mttt yvw"
```

Programming Structures - while loop

The while loop is similar to the for-loop. The first is adds up all the numbers from 0 to n. The second is the for-loop version of the same thing.

```
sum = 0
n = 1000
i = 1
while (i <= n) {
    sum = sum + i
    i = i + 1
}

sum
[1] 500500

sum = 0
n = 1000
for (i in 1:n) {
   sum = sum + 1
}

sum
```

Programming Structures - while loop

Implementing Newton's method to find the square root of a number if easy with a while loop.

```
N = 121     # The number whose square root we wish to estimate
guess = 9   # Our first guess (its not so good but we can improve it)

abs(N - (guess^2))
[1] 40                # Wow. Our first guess isn't so good.
```

Newton comes to the rescue by giving us a formula to improve the guess.

```
guess = 0.5*(N/guess + guess)
[1] 11.22222                   # This better but not quite there yet
abs(N - (guess^2))
[1] 4.933284

guess = 0.5*(N/guess + guess)  # Even better
abs(N - (guess^2))
[1] 0.04840968

guess = 0.5*(N/guess + guess)  # Perfect.
abs(N - (guess^2))
[1] 4.84e-06
guess
[1] 11
```

Programming Structures - while loop

Implementing Newton's method to find the square root of a number if easy with a while loop.

```
n = 121

iterations = 1

guess = 9

tolerance = 0.0001

diff = n-(guess^2)

while (abs(diff) >= 0.001) {
    cat("Iteration number ",iterations,"\n")
    guess = (n/guess + guess)/2
    diff = n-(guess^2)
    iterations = iterations + 1
}
Iteration number  1
Iteration number  2
Iteration number  3

guess
[1] 11
```