Quantitative Genomics and Genetics, Spring 2018

Computer Lab 1

Announcements
- CMS has not been set up yet for students at the NYC campus.

What is R?

- R is a programming language: A set of commands in a syntax that can be interpreted by a computer to make use of the processors to do computations. A way to tell the computer what to do.
- R is a "scripting" type language: The syntax is interpreted directly to do computations and much of the complex tasks one would like to accomplish are implemented automatically.
- R is specifically developed for doing statistical tasks and computations.

What is R studio?

User friendly interface for R. Makes your life a lot easier.

Key features:

- Syntax highlighting
- Code completion
- Smart indentation
- Tools to manage plots
- Browse files and directories
- Visualizing object structures
- When you run Rstudio it automatically opens an R session
- Now even including git functionality

Why R?

- It's free. (1 year SPSS licences range from $100 (student pricing) to $14,000+)
- R is supported by a large community (constantly getting updated, help communities, new features)
- R is user friendly and easy to learn and provides a foundation for learning more complex languages
- R is particularly good for accomplishing statistical applications
- R has nice graphical outputs (ggplot)
- R is particularly useful in bioinformatics (the bioconductor community has many packages available for commonly used analyses)

Why are we interested in using R?

- We can analyze large datasets in a reproducible way. (You will never, or hardly ever, use excel again)
- We can implement algorithms or statistical analyses that are not available as published functions
- We can automate a complex analysis by putting together multiple steps
- We can simulate data

A few remarks on programming

- It is easy ... and hard.

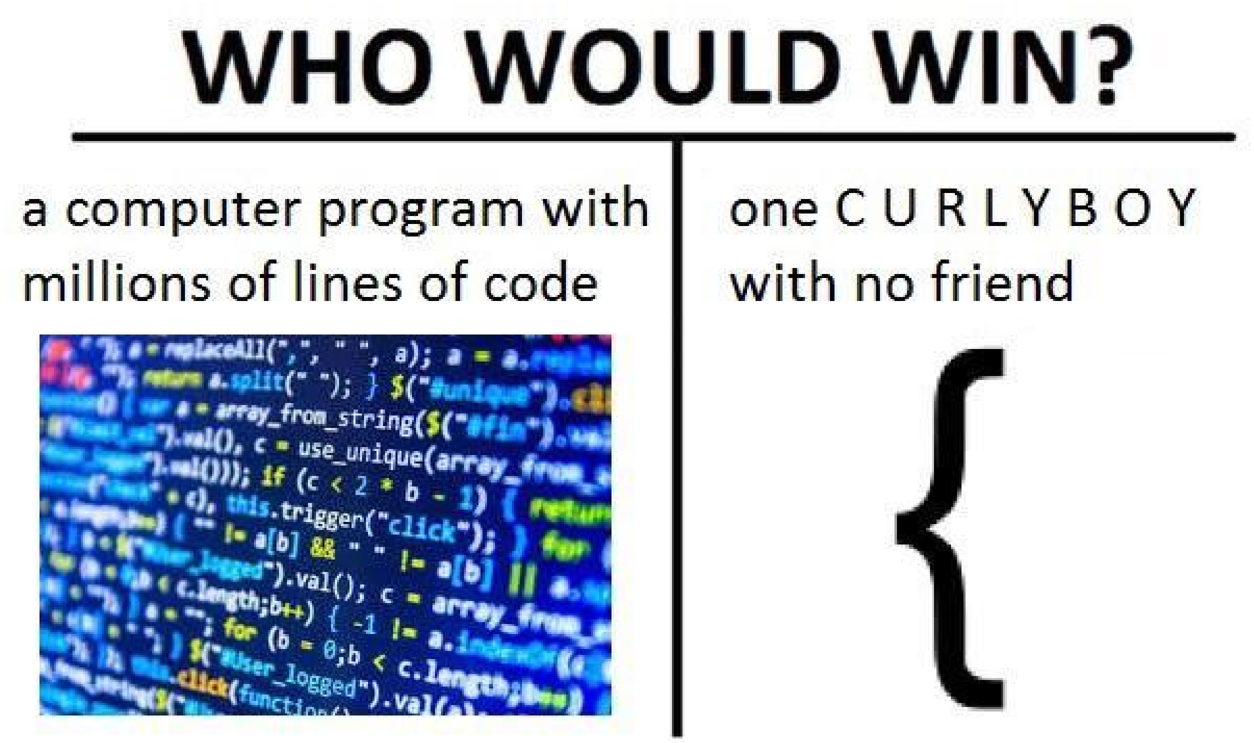
WHO WOULD WIN?
a computer program with millions of lines of code
one C U R L Y B O Y with no friend
{

Figure 1: A good meme

- It is about winning small battles. **Do not attempt to write up the entire process at once**.
- When in doubt, **HIT TAB**.
- Typos are your worst enemy and you will see many in the lecture notes as well…
- Your program or script will almost certainly never work on the first run. If it does work, it is probably not doing the right thing.

1. Location, Location, Location …

The working directory

- The "working directory" is where R automatically looks for your data files and saves any outputs.
- You can use R interactively by writing commands directly to the "prompt", which looks like this ">".
- You can also use the script window to directly run commands. (we will get there soon)
- Let's check the directory that you are in first

```r
getwd()
```

- Now lets set your working directory to another place
- If you don't know what to type as your directory, **try hitting the tab button inside the quote marks**. setwd("-hit tab-")

```r
setwd("Path_to_directory")
```

- This command changed our working directory and we can check it by using getwd() again

```r
getwd()
```

- To see what is in the working directory

```r
dir()
```

```
 [1] "computerLab1.pdf" "iris"             "iris.jpg"
 [4] "iris.RData"       "iris.RDS"         "lab1_data.csv"
 [7] "lab1.Rmd"         "meme.jpg"         "myLab1.html"
[10] "myLab1.Rmd"       "myRmd.zip"
```

```r
list.files()  # does the same thing
```

```
 [1] "computerLab1.pdf" "iris"             "iris.jpg"
 [4] "iris.RData"       "iris.RDS"         "lab1_data.csv"
 [7] "lab1.Rmd"         "meme.jpg"         "myLab1.html"
[10] "myLab1.Rmd"       "myRmd.zip"
```

- Now that we know where we are let's see what R can really do.

2. R as a calculator

```r
101+127
```

```
[1] 228
```

```r
2 * 4
```

```
[1] 8
```

```r
6 / 3
```

```
[1] 2
```

- And in a more sophisticated manner

```r
( (6 / 3) + ( 9 / (1+2) ) - 2.3 )^2
```

```
[1] 7.29
```

- Built in math functions

Side note: Everything that is written after a # is considered as a comment and will not be executed.

```r
log(10)     # log with base e
```

```
[1] 2.302585
```

```r
log2(8)     # log with base 2
```

```
[1] 3
```

```r
log10(1000) # log with base 10
```

```
[1] 3
```

Figure 2: An iris

```r
exp(4)      # exponentials
```

```
[1] 54.59815
```

```r
sqrt(36)    # square roots
```

```
[1] 6
```

```r
abs( 10 - 15 ) # absolute values
```

```
[1] 5
```

- In case you cannot remember what a function does, try typing the name of the function with a ? in front of it.

```r
?log
?log10
```

3. Data structures in R

- From here on I'll be teaching with the irsis dataset

```r
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

1. Vectors

- R can store values in variables that can be declared and used as follows:
- In simpler terms, a vector is just many numbers together, called as one variable

```r
Sepal.Length=iris$Sepal.Length

Sepal.Length
```

```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
```

```
 [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
 [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
 [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8
 [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
 [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
[103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7
[120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7
[137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

- How can I find out how long the vector is, or do other things like mean/variance?
- Use functions! which is functionName(vector)
- Note the parentheses!

```
length(Sepal.Length)
```

```
[1] 150
```

```
mean(Sepal.Length)
```

```
[1] 5.843333
```

```
var(Sepal.Length)
```

```
[1] 0.6856935
```

- What if I want to know how many sepals are longer than 6 in.? Use a comparison and sum

```
Sepal.Length>7
```

```
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[100] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[122] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[133] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
sum(Sepal.Length>7)
```

```
[1] 12
```

- How can I create a new vector of just these 7 in. or greater sepals? Index with a comparison

```
sevenSepals <- Sepal.Length[Sepal.Length>7]
sevenSepals
```

```
 [1] 7.1 7.6 7.3 7.2 7.7 7.7 7.7 7.2 7.2 7.4 7.9 7.7
```

- But what if you get a new measurement? Concatenate the vector with the new measurement
- This is also how you can make a fresh, new vector

```
newMeasurement <- 8.0
Sepal.Length=c(Sepal.Length,newMeasurement)
```

- What if one of your measurements already taken is wrong? Index the wrong measurement and reassign
- Index with []

```
Sepal.Length[1]
```

```
[1] 5.1
```

```
correctMeasurement <- 5.5
Sepal.Length[1] <- correctMeasurement
Sepal.Length[1]
```

```
[1] 5.5
```

-What if you measured in centimeters but want to convert to inches? Multiple the vector

```
CentimeterInInches <- 2.54
Sepal.Length <- Sepal.Length/2.54
```

- What if every iris plant has two sepal lengths and you only want one from each? Index with seq

```
Sepal.Length[seq(1,length(Sepal.Length),2)]
```

```
 [1] 2.165354 1.850394 1.968504 1.811024 1.732283 2.125984 1.889764
 [8] 2.283465 2.125984 2.244094 2.125984 1.811024 1.889764 1.968504
[15] 2.047244 1.889764 2.047244 1.929134 2.165354 1.732283 1.968504
[22] 1.732283 2.007874 2.007874 2.086614 2.755906 2.716535 2.559055
[29] 2.480315 2.598425 1.968504 2.362205 2.204724 2.204724 2.440945
[36] 2.322835 2.480315 2.519685 2.677165 2.362205 2.165354 2.283465
[43] 2.125984 2.637795 2.204724 2.165354 2.283465 2.204724 2.244094
[50] 2.007874 2.480315 2.795276 2.559055 1.929134 2.637795 2.559055
[57] 2.677165 2.283465 2.559055 3.031496 2.716535 3.031496 2.637795
[64] 2.440945 2.519685 2.913386 2.519685 2.401575 2.480315 2.362205
[71] 2.637795 2.283465 2.637795 2.480315 2.440945 3.149606
```

- What if you want to convert every other measurement back to centimters? Index or Recycle

```
Sepal.Length[seq(1,length(Sepal.Length),2)] <- Sepal.Length[seq(1,length(Sepal.Length),2)]/CentimeterInI

Sepal.Length <- Sepal.Length*c(1/CentimeterInInches,1)
```

```
Warning in Sepal.Length * c(1/CentimeterInInches, 1): longer object length
is not a multiple of shorter object length
```

- Can vectors only contain numbers? No! But they must contain all the same type of data

```
Species <- iris$Species
Species[1]
```

```
[1] setosa
Levels: setosa versicolor virginica
```

```
Sepal.Length[1] <- Species[1]
Sepal.Length[1]
```

```
[1] 1
```

-So how will I know if two vectors are the same type? Class()

```
class(Sepal.Length)
```

```
[1] "numeric"
```

```r
class(Species)
```

```
[1] "factor"
```

2. Matrices

- With matrices we can start working in 2 dimensions.

```r
irisMat <- as.matrix(iris[,1:2])
head(irisMat)
```

```
     Sepal.Length Sepal.Width
[1,]          5.1         3.5
[2,]          4.9         3.0
[3,]          4.7         3.2
[4,]          4.6         3.1
[5,]          5.0         3.6
[6,]          5.4         3.9
```

- Matricies are two dimensionals: rows and columns
- How can I pull out just one row or column of the matrix? Access rows with the first position, and the columns with the second position or $

```r
irisMat[1,]
```

```
Sepal.Length  Sepal.Width 
         5.1          3.5 
```

```r
irisMat[,1]
```

```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
 [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
 [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
 [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8
 [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
 [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
[103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7
[120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7
[137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

- How can I count the number of row or columns? nrow() or ncol()

```r
nrow(irisMat)
```

```
[1] 150
```

```r
ncol(irisMat)
```

```
[1] 2
```

-What if you want to change the column or row names? Use colnames() or rownames()

```r
colnames(irisMat) <- c("Col1","Col2")
rownames(irisMat) <- 1:nrow(irisMat)
head(irisMat)
```

```
  Col1 Col2
1  5.1  3.5
2  4.9  3.0
3  4.7  3.2
```

```
4  4.6  3.1
5  5.0  3.6
6  5.4  3.9
```

- What if you like a different matrix shape? t()

```
newIrisMat <- t(irisMat)
head(newIrisMat)
```

```
       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
Col1 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
Col2 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
      18  19  20  21  22  23  24  25 26  27  28  29  30  31  32  33  34
Col1 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8  5 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
Col2 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4  3 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
      35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
Col1 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
Col2 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
      52  53  54  55  56  57  58  59  60 61  62  63  64  65  66  67  68
Col1 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2  5 5.9 6.0 6.1 5.6 6.7 5.6 5.8
Col2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7  2 3.0 2.2 2.9 2.9 3.1 3.0 2.7
      69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
Col1 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
Col2 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
      86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
Col1 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
Col2 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
     103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
Col1 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7
Col2 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
     120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
Col1 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7
Col2 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
     137 138 139 140 141 142 143 144 145 146 147 148 149 150
Col1 6.3 6.4   6 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
Col2 3.4 3.1   3 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

- What if you want to combine matricies? Use cbind or rbind

```
newIrisMat <- rbind(c(1,2),irisMat)
head(newIrisMat)
```

```
  Col1 Col2
   1.0  2.0
1  5.1  3.5
2  4.9  3.0
3  4.7  3.2
4  4.6  3.1
5  5.0  3.6
```

```
newIrisMat <- cbind(rep(1,nrow(irisMat)),irisMat)
head(newIrisMat)
```

```
    Col1 Col2
1 1  5.1  3.5
2 1  4.9  3.0
3 1  4.7  3.2
```

```
4 1  4.6  3.1
5 1  5.0  3.6
6 1  5.4  3.9
```

- Warning, you still cannot mix data types!

```
newIrisMat <- rbind(c("a","b"),irisMat)
head(newIrisMat)
```

```
  Col1  Col2
  "a"   "b"
1 "5.1" "3.5"
2 "4.9" "3"
3 "4.7" "3.2"
4 "4.6" "3.1"
5 "5"   "3.6"
```

3. Dataframes

- To deal with multiple types of data in a single object we use data frames
- How can I make my matrix a data frame? as.data.frame()

```
irisMat <- as.matrix(iris[,1:2])
irisDf <- as.data.frame(irisMat)
head(irisDf)
```

```
  Sepal.Length Sepal.Width
1          5.1         3.5
2          4.9         3.0
3          4.7         3.2
4          4.6         3.1
5          5.0         3.6
6          5.4         3.9
```

- Are you sure you can add mixed data types? Yes!

```
irisDf[,3]=rep("A",nrow(irisDf))
head(irisDf)
```

```
  Sepal.Length Sepal.Width V3
1          5.1         3.5  A
2          4.9         3.0  A
3          4.7         3.2  A
4          4.6         3.1  A
5          5.0         3.6  A
6          5.4         3.9  A
```

- Is there a new way to access columns? Yes! use the $

```
head(irisDf$Sepal.Width)
```

```
[1] 3.5 3.0 3.2 3.1 3.6 3.9
```

- Is there a quick way to check all of the column data types? str()

```
str(irisDf)
```

```
'data.frame':   150 obs. of  3 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ V3          : chr  "A" "A" "A" "A" ...
```

- Is there a quick way to get the stats of each column? summary()

```r
summary(irisDf)
```

```
  Sepal.Length    Sepal.Width         V3           
 Min.   :4.300   Min.   :2.000   Length:150        
 1st Qu.:5.100   1st Qu.:2.800   Class :character  
 Median :5.800   Median :3.000   Mode  :character  
 Mean   :5.843   Mean   :3.057                     
 3rd Qu.:6.400   3rd Qu.:3.300                     
 Max.   :7.900   Max.   :4.400                     
```

- Is there a boring excel type way to see the data? View()

```r
#View(irisDf)
```

4. Lists

- I personally do not like lists, as they tend to be unorganized, but they can come in handy
- Lists are basically bundles of objects, holding vectors, matrices and data frames in a single object.
- How can I create a list? list()

```r
irisList <- list("iris",head(Sepal.Length),head(irisMat),head(irisDf))
head(irisList)
```

```
[[1]]
[1] "iris"

[[2]]
[1] 1.0000000 1.9291339 0.2868116 1.8110236 0.3051187 2.1259843

[[3]]
     Sepal.Length Sepal.Width
[1,]          5.1         3.5
[2,]          4.9         3.0
[3,]          4.7         3.2
[4,]          4.6         3.1
[5,]          5.0         3.6
[6,]          5.4         3.9

[[4]]
  Sepal.Length Sepal.Width V3
1          5.1         3.5  A
2          4.9         3.0  A
3          4.7         3.2  A
4          4.6         3.1  A
5          5.0         3.6  A
6          5.4         3.9  A
```

- How can I name the elements within a list? names()

```r
names(irisList)=c("one","two","three","four")
names(irisList)
```

```
[1] "one"   "two"   "three" "four" 
```

- How can I access elements within lists? [[]] or $

```
irisList[[1]]
```

```
[1] "iris"
```

```
irisList$one
```

```
[1] "iris"
```

Quick Note

- In case you cannot remember which variables you have declared, use the ls() function

```
ls()
```

```
 [1] "CentimeterInInches" "correctMeasurement" "irisDf"
 [4] "irisList"           "irisMat"            "newIrisMat"
 [7] "newMeasurement"     "Sepal.Length"       "sevenSepals"
[10] "Species"
```

- To remove variables that are no longer used, use the rm() function

```
rm(irisList)
ls()
```

```
[1] "CentimeterInInches" "correctMeasurement" "irisDf"
[4] "irisMat"            "newIrisMat"         "newMeasurement"
[7] "Sepal.Length"       "sevenSepals"        "Species"
```

4. Importing and Exporting Data

- How can I save my a data frame? write.table()

```
write.table(iris,"iris")
```

- How can I load a data frame? write.table()

```
doubleIris <- read.table("iris")
head(doubleIris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

- How else can I save my data? Many ways:

```
save(iris,file="iris")
saveRDS(iris,file="iris.RDS")
save.image(file = "iris.RData")
```

5. Questions

- Beginner

```
df <- data.frame(a = c(1, 2, 3), b = c(4, 5, 6), c=c(7, 8, 9))
```

1. From df create a vector of 4,5,6
2. From df create a vector of 2
3. Multiply the first row of df by the second column
4. Append the answer to 3 as a new column of df called d
5. Efficiently as possible, in row 1 multiply the first element by 2, second by 3, third by 2, fourth by 3
6. Save this data frame, tab delimiated, with column names to a file called "Lab1"

- Advanced

1. Give an example where <- and = are not the same
2. In one command remove the first row and first column from iris
3. Represent the first column of iris in R code (as you would have to write it if you were going to create the first column)
4. Change the first iris column name to "Weird Name", then subset the column with $
5. Show a nice R function/trick, maybe it will make it into lab next week