R basics and flow control

Michael Alfaro

9/14/2021

Getting started

Welcome to the EEB Quantitative skills bootcamp.

R basics

Use setwd() to set your directory.

```
setwd("~/Dropbox/bootcamp_examples")
```

And use getwd() to see the working directory.

```
setwd("~/Dropbox/bootcamp_examples")
getwd()
```

```
## [1] "/Users/michaelalfaro/Dropbox/bootcamp_examples"
```

---

comments

The # character is used to mark comments. R ignores everything after the # to the end of the line

```
2 + 2
```

```
## [1] 4
```

```
#2 + 3
```

R ignores the line 2 + 3 because of the '#'

---

getting help

R has many options for getting help. You can use the help() function on any function:

```
help(lm)
```

You can also use "?" before a function.

```
?lm
```

Two question marks ("??") tells R to use fuzzy matching on the function name. R will search for functions with names similar to your query in all installed packages. "?" and "??" won't evaluate in this document but you should try them at your command prompt to see the help files.

```
??lm

---
```

the c() function

The c() function combines elements into a vector and is one of the most commonly use functions in R.

```
grad.school.tips <- c( "use a reference manager", "learn a programming language", "write lots of papers
```

You can use cat() to print objects to a screen.

```
cat(grad.school.tips, sep = "\n")
```

```
## use a reference manager
## learn a programming language
## write lots of papers
```

---

install

install() is used to install new packages:

```
install.packages(c("geiger", "picante"), dep = T)
```

---

variables

As you work in an R session, any variables that you declare will be stored in the session. If you want to see all objects that you have created in you session, use the ls() function. In R the assignment operator is `<-` as in

```
xx <-1000
ls()
```

```
## [1] "grad.school.tips" "xx"
```

We can investigate variable types with `class`:

```
class(xx)
```

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

Basic Data Types

- character (strings)
- numeric (real numbers)
- integer (integer numbers)
- complex (complex numbers)
- logical (TRUE, FALSE)
- factor (categorical values)

---

removing variables

To remove a variable from the workspace, use rm(variable)

```
ls()
```

```
## [1] "grad.school.tips" "xx"
```

```
rm(xx)
ls()
```

```
## [1] "grad.school.tips"
```

quitting R

You can quit R with q(). Caution, q() will quit your R session!

```
q()
q(save = 'no')
```

vectors

Vectors are one dimensional and contain values of the same type

```
evens <- c(2,4,6,8)
fifths <- c("C", "G", "D", "A", "E")
class(evens)
```

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

```
class(fifths)
```

```
## [1] "character"
```

you can perform operations on the entire vector. . . .

```
mean(evens)
```

```
## [1] 5
```

```
sum(evens)
```

```
## [1] 20
```

```r
length(fifths)
```

```
## [1] 5
```

---

matrices and arrays

Matrices are two dimensional versions of vectors. They contain data of the same type and can allow for matrix operations.

```r
mm <- matrix(data = 1:9, nrow = 3, ncol = 3, byrow = TRUE)
mm[1,] # first row
```

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

```r
mm[,1] # first col
```

```
## [1] 1 4 7
```

arrays have more than two dimensions

```r
aa <- array(data = 1:27, dim = c(3,3,3))
aa
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   19   22   25
## [2,]   20   23   26
## [3,]   21   24   27
```

---

lists

lists are collections of any kind of R object (vectors, matrices, data frames). You access list elements by `[]` or by the name of the list element.

```r
ll <- list(vec = aa, mat = mm, char = fifths, num = evens)
ll[[1]]
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   19   22   25
## [2,]   20   23   26
## [3,]   21   24   27
```

```
ll$char
```

```
## [1] "C" "G" "D" "A" "E"
```

---

data frames

These are the most common data structure you will interact with in R. Think of them like excel spreadsheets. We

```
L3 <- LETTERS[1:3]
fac <- sample(L3, 10, replace = TRUE)
dd <- data.frame(x = 1, y = 1:10, fac = fac)
head(dd,3)
```

```
##   x y fac
## 1 1 1   B
## 2 1 2   C
## 3 1 3   C
```

```
str(dd)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ x  : num  1 1 1 1 1 1 1 1 1 1
##  $ y  : int  1 2 3 4 5 6 7 8 9 10
##  $ fac: chr  "B" "C" "C" "A" ...
```

```
## The "same" with automatic column names:
```

We will work with data frames in the next example.

Reading in files and manipulating data objects

For this section we are going to work with two kinds of data: a phylogenetic tree, and swimming data for some of the species in this tree. One of the most common tasks you will perform in R will be reading in data and this section should help orient you to ways you can interact with your data objects in the R environment.

Read in the tree

The first thing we will do is use read.tree() to in a phylogenetic tree. readtree() is in the Ape library, so make sure you have that installed. The tree file is a text file that contains informaiton about the tree structure and tip labels in Newick format. Use a text editor to look at this file if you are curious.

```
library(ape)
tt <- read.tree("~/Dropbox/bootcamp_examples/tree.tre")
###see elements of an object
attributes(tt)
```

```
## $names
## [1] "edge"        "edge.length" "Nnode"       "node.label"  "tip.label"
## [6] "root.edge"
##
## $class
## [1] "phylo"
##
## $order
## [1] "cladewise"
```

```
###access those elements with $
tt$tip.label[1:10]
```

```
##  [1] "Polyodon_spathula"           "Psephurus_gladius"
##  [3] "Scaphirhynchus_albus"        "Scaphirhynchus_platorynchus"
##  [5] "Scaphirhynchus_suttkusi"     "Huso_huso"
##  [7] "Acipenser_dabryanus"         "Acipenser_nudiventris"
##  [9] "Acipenser_ruthenus"          "Acipenser_gueldenstaedtii"
```

```
head(tt$tip.label)
```

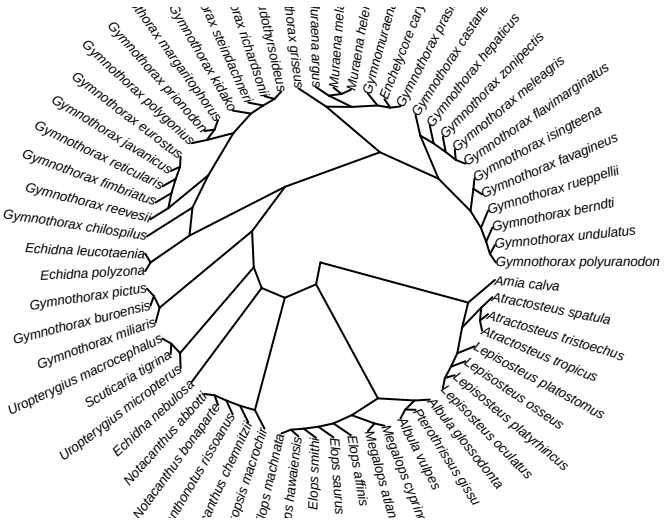
```
## [1] "Polyodon_spathula"           "Psephurus_gladius"
## [3] "Scaphirhynchus_albus"        "Scaphirhynchus_platorynchus"
## [5] "Scaphirhynchus_suttkusi"     "Huso_huso"
```

---

pruning the tree

This tree is giant! Lets prune down and plot the pruned tree.

```
pruned.tree <- drop.tip(tt, tt$tip.label[1:7900])
plot(ladderize(pruned.tree), cex = 0.4, type = "radial")
```



---

reading in data

```
# d contains length data, family, species, order, etc
inpath = "~/Dropbox/bootcamp_examples/data.txt"
dd <- read.table(inpath, header=T, sep='\t', as.is = T);

###NOTE: R by default reads character columns as FACTORS. This data structure behaves very differently
```

---

a quick note about data frames

You have just read your data in as a data frame object. check this with the str() function

```
str(dd)
```

```
## 'data.frame':    92 obs. of  2 variables:
##  $ species: chr  "Naso_brevirostris" "glass_fish" "Zebrasoma_scopas" "Apogon_nigrofasciatus" ...
##  $ mode   : chr  "BCF" NA "BCF" "BCF" ...
```

```
#a data frame is a collection of columns where every object within the column vector is the same data ty
#get the dimensions of a data frame
dim(dd)
```

```
## [1] 92  2
```

```
length.dd <- dim(dd)[1] #what does this line do?
#dimensions are rows, columns
attributes(dd)
```

```
## $names
## [1] "species" "mode"
##
## $class
## [1] "data.frame"
##
```

```
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
```

---

adding data to a data frame

Lets create some size data and add it to the data frame

```
#get 92 random variables
size <- runif(length.dd)

#you can add columns to an existing data frame with cbind
head(dd) #before
```

```
##                          species mode
## 1              Naso_brevirostris  BCF
## 2                     glass_fish <NA>
## 3               Zebrasoma_scopas  BCF
## 4          Apogon_nigrofasciatus  BCF
## 5        Cheilodipterus_macrodon  BCF
## 6 Cheilodipterus_quinquelineatus  BCF
```

```
dd<- cbind(dd, size)
head(dd) #after
```

```
##                          species mode       size
## 1              Naso_brevirostris  BCF 0.05908054
## 2                     glass_fish <NA> 0.33892545
## 3               Zebrasoma_scopas  BCF 0.48657979
## 4          Apogon_nigrofasciatus  BCF 0.54941433
## 5        Cheilodipterus_macrodon  BCF 0.62846143
## 6 Cheilodipterus_quinquelineatus  BCF 0.79519670
```

---

accessing data frame elements

You can use the "$" operator to access rows and head() and tail() check a data frame

```
names(dd) #these are the names of the columns we could access
```

```
## [1] "species" "mode"    "size"
```

```
#dd$species #all the species
head(dd$species)
```

```
## [1] "Naso_brevirostris"              "glass_fish"
## [3] "Zebrasoma_scopas"               "Apogon_nigrofasciatus"
## [5] "Cheilodipterus_macrodon"        "Cheilodipterus_quinquelineatus"
```

```
tail(dd$species) # use these functions to check that data has been read into R correctly
```

```
## [1] "Pomacentrus_coelestis"     "Pomacentrus_lepidogenys"
## [3] "Pomacentrus_nagasakiensis" "Premnas_biaculeatus"
## [5] "Stegastes_apicalis"        "Canthigaster_valentini"
```

```
#you can pull out individual columns
swimming_mode <- dd$mode
```

---

subsetting

use the [] after a data frame to access specific cells, rows, and columns

```
dd[1,1] # entry in row 1, column 1
```

```
## [1] "Naso_brevirostris"
```

```
dd[1,2] # entry in row 1, column 2
```

```
## [1] "BCF"
```

```
dd[1,3] # entry in row 1, column 3
```

```
## [1] 0.05908054
```

```
dd[1,] # row 1, all columns
```

```
##             species mode       size
## 1 Naso_brevirostris  BCF 0.05908054
```

```
dd[,2] # all rows, column 2
```

```
##  [1] "BCF" NA    "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF"
## [13] "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF"
## [25] "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "BCF" "MPF" "MPF" "MPF"
## [37] "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF"
## [49] "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF"
## [61] "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF"
## [73] "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF"
## [85] "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF" "MPF"
```

---

accessing by row name

Naming rows allows you to access a row by name (Note that rownames are a part of a data frame but not a separate column of the data frame)

```
head(rownames(dd))
```

```
## [1] "1" "2" "3" "4" "5" "6"
```

```
rownames(dd) <- dd$species
head(rownames(dd))
```

```
## [1] "Naso_brevirostris"             "glass_fish"
## [3] "Zebrasoma_scopas"              "Apogon_nigrofasciatus"
## [5] "Cheilodipterus_macrodon"       "Cheilodipterus_quinquelineatus"
```

```r
str(dd)
```

```
## 'data.frame':    92 obs. of  3 variables:
##  $ species: chr  "Naso_brevirostris" "glass_fish" "Zebrasoma_scopas" "Apogon_nigrofasciatus" ...
##  $ mode   : chr  "BCF" NA "BCF" "BCF" ...
##  $ size   : num  0.0591 0.3389 0.4866 0.5494 0.6285 ...
```

```r
# if you name the columns you can access a row by name
dd['Pomacentrus_brachialis',]
```

```
##                                       species mode      size
## Pomacentrus_brachialis Pomacentrus_brachialis  MPF 0.4310697
```

A bit more on subsetting

```r
#a bit on subseting
dd[5:10,] # rows 5-10, all columns
```

```
##                                                  species mode      size
## Cheilodipterus_macrodon               Cheilodipterus_macrodon  BCF 0.6284614
## Cheilodipterus_quinquelineatus Cheilodipterus_quinquelineatus  BCF 0.7951967
## Chaetodon_aureofasciatus             Chaetodon_aureofasciatus  BCF 0.7203486
## Chaetodon_auriga                            Chaetodon_auriga  BCF 0.4821063
## Chaetodon_baronessa                      Chaetodon_baronessa  BCF 0.9985122
## Chaetodon_citrinellus                  Chaetodon_citrinellus  BCF 0.3542065
```

```r
dd[5:10,3] # rows 5-10, column 3
```

```
## [1] 0.6284614 0.7951967 0.7203486 0.4821063 0.9985122 0.3542065
```

---

which()

if we store the results of this which() function we can subset the dataframe to include only MPF swimmwers

```r
#if you want only the MPF swimmers, you can use the which() function
which(dd$mode == 'MPF')
```

```
##  [1] 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
## [26] 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
## [51] 84 85 86 87 88 89 90 91 92
```

```r
mpfs <- which(dd$mode == 'MPF') #stores rows of mpf swimmers
mpf_swimmers <- dd[mpfs,] #stored this as a seperate df
head(mpf_swimmers)
```

```
##                                       species mode       size
## Acanthurus_blochii            Acanthurus_blochii  MPF 0.78593403
## Acanthurus_dussumieri      Acanthurus_dussumieri  MPF 0.93140185
## Acanthurus_lineatus          Acanthurus_lineatus  MPF 0.25164487
## Acanthurus_nigrofuscus Acanthurus_nigrofuscus  MPF 0.82633123
## Acanthurus_olivaceus        Acanthurus_olivaceus  MPF 0.86893561
## Acanthurus_triostegus      Acanthurus_triostegus  MPF 0.01812155
```

Writing data

recall that we used `read.table` to read in the original data file. R offer a number of different read commands depending on the format needed including `read.table` (space or tab separated data), `read.csv` (comma separated) and `read.csv2` (semi-colon separated). For writing data there are `writetable`, `write.csv` and `write.csv2` functions. R by default will attach quotes around variable names and will also write out the row names/numbers. If you do not want your file to look like that you can change the write arguments.

```
write.table(mpf_swimmers, "mpfs_r_default.csv") # space separator, quotes, row names
write.csv(mpf_swimmers, "mpfs_r_better_format.csv", quote = F, row.names = F) # comma separated, no quo
```

R Challenge

How would you make dataframe with all of the big (size > 0.9) species only?

```
head(dd)
```

```
##                                                species mode       size
## Naso_brevirostris                        Naso_brevirostris  BCF 0.05908054
## glass_fish                                      glass_fish <NA> 0.33892545
## Zebrasoma_scopas                          Zebrasoma_scopas  BCF 0.48657979
## Apogon_nigrofasciatus                  Apogon_nigrofasciatus  BCF 0.54941433
## Cheilodipterus_macrodon               Cheilodipterus_macrodon  BCF 0.62846143
## Cheilodipterus_quinquelineatus Cheilodipterus_quinquelineatus  BCF 0.79519670
```

```
which(dd$size > 0.9) #shows us rows with large fish in them
```

```
## [1]  9 31 33 35 44 57 73 89
```

R challenge

make a new data frame with large species only

hints

- use the **which()** function to select only rows of some size or greater
- the **$** operator lets you specify columns from a data frame
- you can subset a data frame by specifying a list of row names within the square backets []

---

one solution

```
big.fish <- dd[which(dd$size > 0.9),] #remember the , after the which command says "select all columns"
head(big.fish)
```

```
##                                          species mode      size
## Chaetodon_baronessa                Chaetodon_baronessa  BCF 0.9985122
## Siganus_doliatus                      Siganus_doliatus  BCF 0.9899701
## Siganus_vulpinus                      Siganus_vulpinus  BCF 0.9692997
```

```
## Acanthurus_dussumieri                Acanthurus_dussumieri  MPF 0.9314019
## Bodianus_mesothorax                    Bodianus_mesothorax  MPF 0.9999017
## Macropharyngodon_negrosensis Macropharyngodon_negrosensis  MPF 0.9367839
```

---

checking for NAs

Sometimes your data frame will include missing values. Often you will want to exclude these rows from the analysis. There are several ways to do this.

```
#ways to check for NAs
head(dd) # there are NAs in the data
```

```
##                                                 species mode       size
## Naso_brevirostris                       Naso_brevirostris  BCF 0.05908054
## glass_fish                                     glass_fish <NA> 0.33892545
## Zebrasoma_scopas                         Zebrasoma_scopas  BCF 0.48657979
## Apogon_nigrofasciatus                 Apogon_nigrofasciatus  BCF 0.54941433
## Cheilodipterus_macrodon             Cheilodipterus_macrodon  BCF 0.62846143
## Cheilodipterus_quinquelineatus Cheilodipterus_quinquelineatus  BCF 0.79519670
```

```
head(is.na(dd))
```

```
##                                species  mode  size
## Naso_brevirostris                FALSE FALSE FALSE
## glass_fish                       FALSE  TRUE FALSE
## Zebrasoma_scopas                 FALSE FALSE FALSE
## Apogon_nigrofasciatus            FALSE FALSE FALSE
## Cheilodipterus_macrodon          FALSE FALSE FALSE
## Cheilodipterus_quinquelineatus   FALSE FALSE FALSE
```

```
which(is.na(dd$mode)) #item 2
```

```
## [1] 2
```

```
complete.cases(dd)
```

```
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [85]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

In each case notice that the record for glass_fish has an NA for swimming mode.

---

removing NAs

We can remove these missing cases in a variety of ways.

```
#one way to get only complete cases
cleaned_1 <- dd[complete.cases(dd),]
#another
```

```
cleaned_2 <- na.omit(dd)

dd <- cleaned_1
```

Note that we have reassigned the cleaned data set to dd so that **dd** only includes the complete cases.

Renaming data frame entries and matching data objects

The following example demonstrates how to manipulate data frames from different sources to find elements in common. We will not go through this example in class this year due to constraints on time. However I am including it as a optional exercise to work through for those of you who might find it useful.

You will often need to find common elements between two data sets before you can do an analysis of the data. In our example we have a phylogeny that is taken from one study and a data set on swimming mode that is taken from another. Problems:

- tree huge
- data may not match species in tree

---

setdiff()

setdiff() is a useful tool. setdiff() compares two lists and returns the items in the first list that are not present in the second list. Also see intersect(), union(), and setdiff().

```
setdiff(dd$species, tt$tip.label)
```

```
##  [1] "Apogon_nigrofasciatus"          "Cheilodipterus_quinquelineatus"
##  [3] "Chaetodon_lunulatus"            "Chaetodon_plebius"
##  [5] "Chaetodon_rainfordii"           "Heniochus_singularis"
##  [7] "Amblygobius_decussatus"         "Scolopsis_bilineatus"
##  [9] "Acanthurus_lineatus"            "Sufflamen_chrysopterus"
## [11] "Cheilinus_chlorurus"            "Cirrhilabrus_punctatus"
## [13] "Oxycheilinus_digrammus"         "Pseudocheilinus_hexataenia"
## [15] "Thalassoma_janseni"             "Chrysiptera_brownriggi"
## [17] "Neoglyphidodon_melas?"
```

Changing one field in a record

OK, it looks like there are 18 species in the swimming data set that don't match the tree. Some of these mismatches are due to spelling errors or taxonomic inconsistency between the two data sets. Here is one way we could correct a name.

```
dd$species[which(dd$species == 'Chaetodon_plebius')]<-'Chaetodon_plebeius' #taxonomic inconsistency
```

---

matching rest of data to tree

```
del_from_data <- setdiff(dd$species, tt$tip.label)
# tips with data not in tree
del_from_data
```

```
##  [1] "Apogon_nigrofasciatus"          "Cheilodipterus_quinquelineatus"
##  [3] "Chaetodon_lunulatus"            "Chaetodon_rainfordii"
##  [5] "Heniochus_singularis"           "Amblygobius_decussatus"
##  [7] "Scolopsis_bilineatus"           "Acanthurus_lineatus"
##  [9] "Sufflamen_chrysopterus"         "Cheilinus_chlorurus"
## [11] "Cirrhilabrus_punctatus"         "Oxycheilinus_digrammus"
## [13] "Pseudocheilinus_hexataenia"     "Thalassoma_janseni"
## [15] "Chrysiptera_brownriggi"         "Neoglyphidodon_melas?"
```

```
#keep all species in data file except those that match the del_from_data
pruned_data <- dd[!(dd$species %in% del_from_data),]

setdiff(pruned_data$species, tt$tip.label) # this should produce "character(0)" if empty.
```

```
## character(0)
```

---

matching tree to data

Now we've pruned the data set. How can figure out what tips of the tree to prune? settdiff() again, but this time swtching the order of the arguments

```
not.in.dd <-setdiff(tt$tip.label, pruned_data$species )
length(not.in.dd) #this will be a large number because the tree has so many tips!
```

```
## [1] 7888
```

```
head(not.in.dd)
```

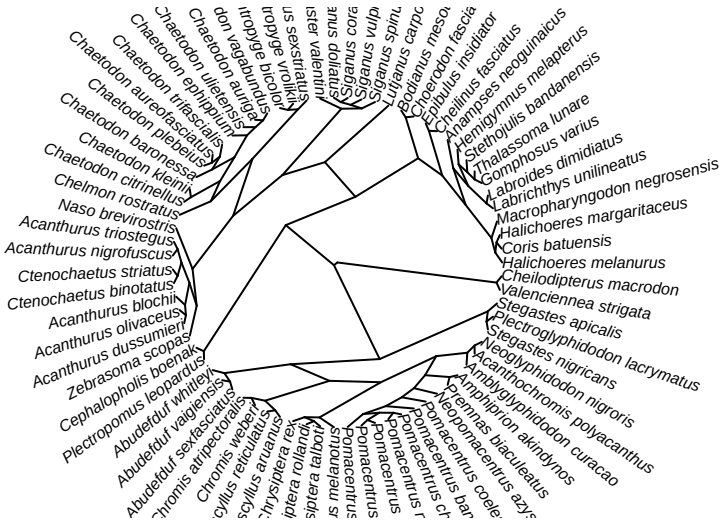
```
## [1] "Polyodon_spathula"           "Psephurus_gladius"
## [3] "Scaphirhynchus_albus"        "Scaphirhynchus_platorynchus"
## [5] "Scaphirhynchus_suttkusi"     "Huso_huso"
```

Now we will use the drop.tip() function from ape to any tip that is in not.in.dd. drop.tip() needs a tree and a list of taxa to be dropped as arguments and returns a pruned tree. Use the help function to verify this.

```
pruned.tree <- drop.tip(tt, not.in.dd)
setdiff(pruned.tree$tip.label, pruned_data$species) #should be "character 0" if these objects match
```

```
## character(0)
```

```
plot(pruned.tree, type = "radial",cex = 0.5)
```



We now have a tree and matching data set. You can use setdiff() and match() as shown above to compare your own data files and prune them as needed.

Introduction to control statements

Control statements order operations

- **for** each line in a text file
  - capitalize the first word
- **while** the number of simulations is less than 100:
  - perform new simulation
- **if** the sample is from Cuba, Hispaniola, or Jamaica:
  - assign sample to "island"
- **else**
  -

assign sample as mainland

More help with data manipulation in R

We have only scratched the surface in exploring the power of R to wrangle data. For more practice and guided tutorials on R data fundamentals check out the Software Carpentry site here. In addition there is a popular and powerful set of tools for working with data wrapped up in a set of packages called the tidyverse. This is my preferred way for working with data now. If you want to learn the tidyverse tools you can start at this UCLA tidyverse course site and here and here. Please also let me know if you find sites that are especailly helpful and I will share them with the class.

Common control statements

for statements perform an action over a range **for (some range) {do something}

```
for (ii in 1:5){
  cat("\nthe number is ", ii)
}
```

```
##
## the number is  1
## the number is  2
## the number is  3
## the number is  4
## the number is  5
```

---

more about for

You can also loop over all items in a vector

```
notfish <- c("bat", "dolphin", "toad", "soldier")

for(animal in notfish){
  cat(animal, "fish\n", sep="")
}
```

```
## batfish
## dolphinfish
## toadfish
## soldierfish
```

while loops keeps running until the conditional part of the expression fails. At this point, the loop is terminated.

```
thesis_idea_sucks <- True #initialize ideas to suck

while(thesis_idea_sucks){
    current_idea <- get_New_Thesis_Idea();


}
```

When a good idea is returned, the program breaks out of the loop.

if

if statements allow your code to diverge depending on conditions

IF (condition is true) {do something}

```
if (xx == 'a') doSomething1;
if (xx == 'b') doSomething2;
if (xx=='c') doSomething3;
```

else if

Use **else if** with **if** and **else** when you have multiple conditions

```
if (x == 'a'){
  doSomething1;
}
else if (x == 'b'){
```

```
    doSomething2;
}...
else if (x == 'z'){
    doSomething26;
}
else{
cat('x != a letter\n')
}
```

Pseudocode

Pseudocode is an informal way to plan out the structure and flow of your program.

- don't worry about syntax of a particular language
- **do** think about variables and control structure
- Pseudocode can be translated across many languages easily

Pseudocode example

Write a script that prints a number and its square over a given range on integers and then sums them.

```
# set lower and upper range values
# set squaresum to 0

# loop over the range and for each value print
  # currentvalue and the currentvalue^2
  # add currentvalue^2 to squaresum
# print "here is the sum of it all"m squaresum
```

Try this now!

---

one solution

```
lower = 1; upper = 5; squaresum = 0

for (ii in lower:upper){
  cat(ii, ii^2, "\n")
  squaresum <- squaresum +  ii^2
}
```

```
## 1 1
## 2 4
## 3 9
## 4 16
## 5 25
```

```
cat("the sum of it all is ", squaresum)
```

```
## the sum of it all is  55
```

---

functions

A function is a self-contained bit of code that performs a task. It might sum a set of numbers, run a simulation, or print your name backwards 500 times.

Functions are useful because

- they make code modular
- they make code reuseable
- they isolate code from unintended consequences (**scope**)

---

how functions work

Usually functions...

- take one or more arguments
- perform some operations
- return something

```
doubler <- function(num){
  ## this function takes a number and doubles it
  doubled = 2 * num
  cat("witness the awesome power of the doubler\n")
  cat("I changed ", num, " to ", doubled, "\n")
  cat("you're welcome!\n")
  return(doubled)
}

for (ii in 1:100){
  doubled <- doubler(ii)
  cat(doubled, "\n")
}
```

```
## witness the awesome power of the doubler
## I changed  1  to  2
## you're welcome!
## 2
## witness the awesome power of the doubler
## I changed  2  to  4
## you're welcome!
## 4
## witness the awesome power of the doubler
## I changed  3  to  6
## you're welcome!
```

```
## 6
## witness the awesome power of the doubler
## I changed  4  to  8
## you're welcome!
## 8
## witness the awesome power of the doubler
## I changed  5  to  10
## you're welcome!
## 10
## witness the awesome power of the doubler
## I changed  6  to  12
## you're welcome!
## 12
## witness the awesome power of the doubler
## I changed  7  to  14
## you're welcome!
## 14
## witness the awesome power of the doubler
## I changed  8  to  16
## you're welcome!
## 16
## witness the awesome power of the doubler
## I changed  9  to  18
## you're welcome!
## 18
## witness the awesome power of the doubler
## I changed  10  to  20
## you're welcome!
## 20
## witness the awesome power of the doubler
## I changed  11  to  22
## you're welcome!
## 22
## witness the awesome power of the doubler
## I changed  12  to  24
## you're welcome!
## 24
## witness the awesome power of the doubler
## I changed  13  to  26
## you're welcome!
## 26
## witness the awesome power of the doubler
## I changed  14  to  28
## you're welcome!
## 28
## witness the awesome power of the doubler
## I changed  15  to  30
## you're welcome!
## 30
## witness the awesome power of the doubler
## I changed  16  to  32
## you're welcome!
## 32
## witness the awesome power of the doubler
```

```
## I changed  17  to  34
## you're welcome!
## 34
## witness the awesome power of the doubler
## I changed  18  to  36
## you're welcome!
## 36
## witness the awesome power of the doubler
## I changed  19  to  38
## you're welcome!
## 38
## witness the awesome power of the doubler
## I changed  20  to  40
## you're welcome!
## 40
## witness the awesome power of the doubler
## I changed  21  to  42
## you're welcome!
## 42
## witness the awesome power of the doubler
## I changed  22  to  44
## you're welcome!
## 44
## witness the awesome power of the doubler
## I changed  23  to  46
## you're welcome!
## 46
## witness the awesome power of the doubler
## I changed  24  to  48
## you're welcome!
## 48
## witness the awesome power of the doubler
## I changed  25  to  50
## you're welcome!
## 50
## witness the awesome power of the doubler
## I changed  26  to  52
## you're welcome!
## 52
## witness the awesome power of the doubler
## I changed  27  to  54
## you're welcome!
## 54
## witness the awesome power of the doubler
## I changed  28  to  56
## you're welcome!
## 56
## witness the awesome power of the doubler
## I changed  29  to  58
## you're welcome!
## 58
## witness the awesome power of the doubler
## I changed  30  to  60
## you're welcome!
```

```
## 60
## witness the awesome power of the doubler
## I changed  31  to  62
## you're welcome!
## 62
## witness the awesome power of the doubler
## I changed  32  to  64
## you're welcome!
## 64
## witness the awesome power of the doubler
## I changed  33  to  66
## you're welcome!
## 66
## witness the awesome power of the doubler
## I changed  34  to  68
## you're welcome!
## 68
## witness the awesome power of the doubler
## I changed  35  to  70
## you're welcome!
## 70
## witness the awesome power of the doubler
## I changed  36  to  72
## you're welcome!
## 72
## witness the awesome power of the doubler
## I changed  37  to  74
## you're welcome!
## 74
## witness the awesome power of the doubler
## I changed  38  to  76
## you're welcome!
## 76
## witness the awesome power of the doubler
## I changed  39  to  78
## you're welcome!
## 78
## witness the awesome power of the doubler
## I changed  40  to  80
## you're welcome!
## 80
## witness the awesome power of the doubler
## I changed  41  to  82
## you're welcome!
## 82
## witness the awesome power of the doubler
## I changed  42  to  84
## you're welcome!
## 84
## witness the awesome power of the doubler
## I changed  43  to  86
## you're welcome!
## 86
## witness the awesome power of the doubler
```

```
## I changed  44  to  88
## you're welcome!
## 88
## witness the awesome power of the doubler
## I changed  45  to  90
## you're welcome!
## 90
## witness the awesome power of the doubler
## I changed  46  to  92
## you're welcome!
## 92
## witness the awesome power of the doubler
## I changed  47  to  94
## you're welcome!
## 94
## witness the awesome power of the doubler
## I changed  48  to  96
## you're welcome!
## 96
## witness the awesome power of the doubler
## I changed  49  to  98
## you're welcome!
## 98
## witness the awesome power of the doubler
## I changed  50  to  100
## you're welcome!
## 100
## witness the awesome power of the doubler
## I changed  51  to  102
## you're welcome!
## 102
## witness the awesome power of the doubler
## I changed  52  to  104
## you're welcome!
## 104
## witness the awesome power of the doubler
## I changed  53  to  106
## you're welcome!
## 106
## witness the awesome power of the doubler
## I changed  54  to  108
## you're welcome!
## 108
## witness the awesome power of the doubler
## I changed  55  to  110
## you're welcome!
## 110
## witness the awesome power of the doubler
## I changed  56  to  112
## you're welcome!
## 112
## witness the awesome power of the doubler
## I changed  57  to  114
## you're welcome!
```

```
## 114
## witness the awesome power of the doubler
## I changed  58  to  116
## you're welcome!
## 116
## witness the awesome power of the doubler
## I changed  59  to  118
## you're welcome!
## 118
## witness the awesome power of the doubler
## I changed  60  to  120
## you're welcome!
## 120
## witness the awesome power of the doubler
## I changed  61  to  122
## you're welcome!
## 122
## witness the awesome power of the doubler
## I changed  62  to  124
## you're welcome!
## 124
## witness the awesome power of the doubler
## I changed  63  to  126
## you're welcome!
## 126
## witness the awesome power of the doubler
## I changed  64  to  128
## you're welcome!
## 128
## witness the awesome power of the doubler
## I changed  65  to  130
## you're welcome!
## 130
## witness the awesome power of the doubler
## I changed  66  to  132
## you're welcome!
## 132
## witness the awesome power of the doubler
## I changed  67  to  134
## you're welcome!
## 134
## witness the awesome power of the doubler
## I changed  68  to  136
## you're welcome!
## 136
## witness the awesome power of the doubler
## I changed  69  to  138
## you're welcome!
## 138
## witness the awesome power of the doubler
## I changed  70  to  140
## you're welcome!
## 140
## witness the awesome power of the doubler
```

```
## I changed  71  to  142
## you're welcome!
## 142
## witness the awesome power of the doubler
## I changed  72  to  144
## you're welcome!
## 144
## witness the awesome power of the doubler
## I changed  73  to  146
## you're welcome!
## 146
## witness the awesome power of the doubler
## I changed  74  to  148
## you're welcome!
## 148
## witness the awesome power of the doubler
## I changed  75  to  150
## you're welcome!
## 150
## witness the awesome power of the doubler
## I changed  76  to  152
## you're welcome!
## 152
## witness the awesome power of the doubler
## I changed  77  to  154
## you're welcome!
## 154
## witness the awesome power of the doubler
## I changed  78  to  156
## you're welcome!
## 156
## witness the awesome power of the doubler
## I changed  79  to  158
## you're welcome!
## 158
## witness the awesome power of the doubler
## I changed  80  to  160
## you're welcome!
## 160
## witness the awesome power of the doubler
## I changed  81  to  162
## you're welcome!
## 162
## witness the awesome power of the doubler
## I changed  82  to  164
## you're welcome!
## 164
## witness the awesome power of the doubler
## I changed  83  to  166
## you're welcome!
## 166
## witness the awesome power of the doubler
## I changed  84  to  168
## you're welcome!
```

```
## 168
## witness the awesome power of the doubler
## I changed  85  to  170
## you're welcome!
## 170
## witness the awesome power of the doubler
## I changed  86  to  172
## you're welcome!
## 172
## witness the awesome power of the doubler
## I changed  87  to  174
## you're welcome!
## 174
## witness the awesome power of the doubler
## I changed  88  to  176
## you're welcome!
## 176
## witness the awesome power of the doubler
## I changed  89  to  178
## you're welcome!
## 178
## witness the awesome power of the doubler
## I changed  90  to  180
## you're welcome!
## 180
## witness the awesome power of the doubler
## I changed  91  to  182
## you're welcome!
## 182
## witness the awesome power of the doubler
## I changed  92  to  184
## you're welcome!
## 184
## witness the awesome power of the doubler
## I changed  93  to  186
## you're welcome!
## 186
## witness the awesome power of the doubler
## I changed  94  to  188
## you're welcome!
## 188
## witness the awesome power of the doubler
## I changed  95  to  190
## you're welcome!
## 190
## witness the awesome power of the doubler
## I changed  96  to  192
## you're welcome!
## 192
## witness the awesome power of the doubler
## I changed  97  to  194
## you're welcome!
## 194
## witness the awesome power of the doubler
```

```
## I changed  98  to  196
## you're welcome!
## 196
## witness the awesome power of the doubler
## I changed  99  to  198
## you're welcome!
## 198
## witness the awesome power of the doubler
## I changed  100  to  200
## you're welcome!
## 200
```

---

functions don't need to return anything

```
takeNoArguments <- function() {
cat('this function takes no arguments\n'); cat('it also\n');
cat('returns nothing\n');
cat('you never get something for nothing.\n')
}
takeNoArguments()
```

```
## this function takes no arguments
## it also
## returns nothing
## you never get something for nothing.
```

---

creating functions

To define a function, you use the function keyword like this:

```
myFunction <- function(arg1, arg2)
```

This says that you want you create a function named 'myFunction' which takes two arguments, arg1 and arg2. Below this line, you enclose the statements belonging to the function in curly braces:

```
{
   cat('this is my function');
   cat('dont mess with it');
}
```

---

using functions

Once you have defined your function, it is part of your workspace. Until you remove it, you can use it. Enter the following function:

```
greeter <- function(name) {
  cat('Hello, ', name, '\n');
}
```

greeter() takes the variable **name** as an argument and performs the greeting.

- what happens if you fail to supply argument name?
- what happens if you just type the name of the function without any parentheses?

Further function help

Of course there are a large number of web resoures to help you learn various aspects of R programming. I like the Software Carpentry site here. Please send me other sites you find useful and I can share them with the class.