R programming techniques

4

Basic R 'Built-in' functions for working with objects

- R has many built-in functions for doing simple calculations on objects. Start with a random sample of 15 numbers from 1 to 100 and try the functions below.

```
> x<-sample(100,15)
```

- Arithmetic with vectors
  - Min / Max value number in a series

`min(x) ; max(x)`
  - Sum of values in a series

`sum(x)`
  - Average estimates (mean / median)

`mean(x) ; median(x)`
  - Range of values in a series

`range(x)`
  - Variance

`var(x)`

- Arithmetic with vectors
  - Rank ordering

`rank(x)`
  - Quantiles

•`quantile(x); boxplot(x)`
  - Square Root

`sqrt(x)`
  - Standard deviation

`sd(x)`
  - Trigonometry functions

`tan(x) ; cos(x) ; sin(x)`

Basic R 'Built-in' functions for working with variables

- list & remove objects

```
ls(), rm()
rm(list=ls()) # get rid of everything
```

- Add rows or columns to a data frame, *df*. Row bind, column bind

```
rbind(df,...), cbind(df,...)
```

- Remove a row, or column, from a data frame.

```
df[-1,] # remove first row
df[,-1] # remove first column
```

- Names of objects

```
names(…)
colnames(...)
rownames(...)
```

- Return length of an object, number of rows or columns of a dataframe or matrix

```
length(…)
nrow(…)
ncol(…)
```

Sorting a vector with **sort**:

```
sort(patients$Second_Name)
[1] "Baker"   "Daniels" "Davis"   "Edwards" "Evans"   "Jones"   "Parker"  "Roberts" "Smith"
 "Wilson"
```

Sorting a data frame by one variable with **order**:

```
order(patients$Second_Name)
[1]  5  6  4  7  3  1  2  9  8 10
patients[order(patients$Second_Name),]
```

Looping - informal introduction

- What if we had 100 data files to load in, and we wanted to load them all into one data frame?
- We could do this:

```
> colony<-data.frame()     # Start with empty data frame
> colony<-rbind(colony, read.csv("11_CFA_Run1Counts.csv"))
> colony<-rbind(colony, read.csv("11_CFA_Run2Counts.csv"))
> colony<-rbind(colony, read.csv("11_CFA_Run3Counts.csv"))
...
> colony<-rbind(colony, read.csv("11_CFA_Run100Counts.csv"))
```

But this will be boring to type, difficult to change, and prone to error.

- As we are doing the same thing 100 times, but with a different file name each time, we can use a **loop** instead.

R language elements
Commands & flow control

- Looping
  - Iterate over a set of values (**for** loop)
  - or while a condition is met (**while** loop)

• Loops are very common in most programming languages, but are not as common in R. Because R can do vectorized calculations, there is no need to use loops to do most things – for example, to sum two vectors.

• Loops are multi-line commands. R will execute them only after the whole loop has been typed in. Use Rstudio editor to type it all in, don't do it in R console!

LOOPS
Commands & flow control

- We can generate a filename using **paste**:

```
paste("11_CFA_Run",1,"Counts.csv",sep="")
[1] "11_CFA_Run1Counts.csv"
```

- So we can load all the files using a **for** loop as follows:

```
colony<-data.frame()
for (f in 1:100) {
   t<-read.csv(paste("11_CFA_Run",f,"Counts.csv",sep=""))
   colony<-rbind(colony,t)
}
```

- Or we could use a **while** loop:

```
f <- 1
colony<-data.frame()
while ( f <= 100 ) {
   t<-read.csv(paste("11_CFA_Run",f,"Counts.csv",sep=""))
   colony<-rbind(colony,t)
   f <- f + 1
}
```

when this condition is false the loop stops

Loops with breaks
Commands & flow control

Suppose, for testing purposes, we only wanted to load the first 2 files in, to make sure our analysis worked on those before we load all the data in. We can use an **if** statement to check for a condition:

```
colony<-data.frame()
for (f in 1:100) {
    if (f<=2) {
        t<-read.csv(paste("11_CFA_Run",f,"Counts.csv",sep=""))
        colony<-rbind(colony,t)
    } else {
        warning(paste("Not loading past file ", f))
        break
    }
}
```

The break statement ends the loop on whichever iteration has been reached. The **warning** function prints out an error message, but carries on with the program (use **stop** if you want to output an error and quit).

Conditional branching
Commands & flow control

- Use an **if** statement for any kind of condition testing.
- Different outcomes can be selected based on a condition within brackets.

```
if (condition) {
… do this …
} else {
… do something else …
}
```

  - condition is any logical value, and can contain multiple conditions
    - e.g. `(a==2 & b <5)`, this is a compound conditional argument

Code formatting avoids bugs!

- Code formatting is crucial for readability of loops

```
f<-26
while(f!=0){
print(letters[f])
f<-f-1 }
```

BAD!!!

```
f <- 26
while( f != 0 ){
   print(letters[f])
   f <- f-1
}
```

GOOD!

- The code between brackets {} **always** is indented, this clearly separates what is executed once, and what is run multiple times
- Trailing bracket } always alone on the line at the same indentation level as the initial bracket {
- Use white spaces to divide the horizontal space between units of your code, e.g. around assignments, comparisons

Exercise

1. Load in the **colony** data frame using a for loop. Three of the data files (but not the other 97!) are in the *Day_1_scripts* folder. Load all three files into **colony**.

2. How many observations do you have? Find out by counting the number of rows in **colony** using the **nrow** function.

3. You have calculated that you will have sufficient power for your analysis if you have at least 70 observations. Write a **while** loop that will continue to load files until you have loaded at least 70 observations into the **colony** data frame.

Answers to exercise

1. To load all three files, use the code from the first **for** loop slide, but only specify three files:

```
colony<-data.frame()
for (f in 1:3) {
   t<-read.csv(paste("11_CFA_Run",f,"Counts.csv",sep=""))
   colony<-rbind(colony,t)
}
```

2. Loading enough files to load 70 observations:

```
f <- 1
colony<-data.frame()
while ( nrow(colony)<=70 ) {
   t<-read.csv(paste("11_CFA_Run",f,"Counts.csv",sep=""))
   colony<-rbind(colony,t)
   f <- f + 1
}
```