University of New Hampshire

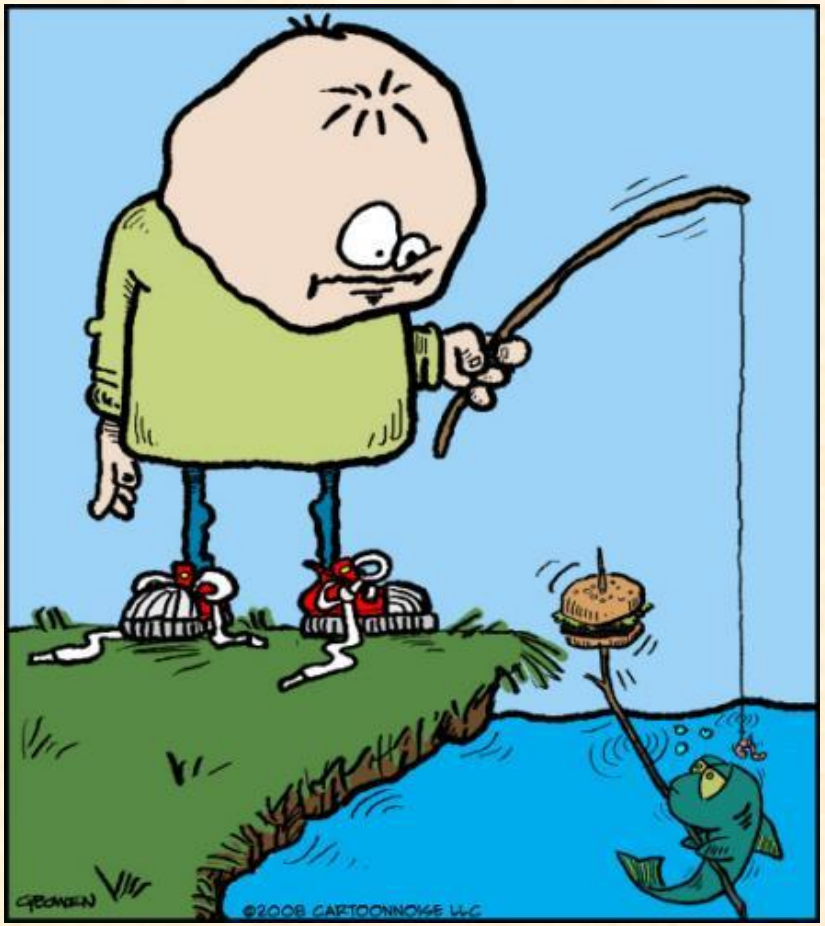
Improving coding and workflow: loops, functions, and more...

R Coding Club Presentation , April 1, 2019

Ernst Linder

Dept. of Math & Statistics
M321B Kingsbury elinder@unh.edu

R-fooling around



Tests with `if` and `switch`

Basic syntax:

```
if(test){
...do if test is true...
} else {
...do if test is false...
}
```

Example (note: { …} are not needed if there is only one command)

```
a = 20
if (a < 20) {
    cat("a is less than 20")
    }else  {
    cat("a is not less than 20")
    }

if (a < 20) cat("a is less than 20") else cat("a is not less than 20")
```

The **switch** function has the following general form.

```
switch(object,
    "value1" = {expr1},
    "value2" = {expr2},
    "value3" = {expr3},
               {other expressions}
)
```

Example (note: { …} are not needed if there is only one command)

```
## choose one x of the following
x = 4 ;  x = 0  ;  x = -4
switch(as.character(sign(x)), "1" = sqrt(x),
    "-1" = print(sqrt(-x)), "0" = cat("x is zero"))
```

Looping with `for`, `while` and `repeat`

for

```
for (i in for_object)
{
  some expressions
}
```

In the for loop some expressions are evaluated for each element i in for object.

while

```
while (condition)
{
   some expressions
}
```

In the while loop some expressions are repeatedly executed until the logical condition is FALSE. Make sure that the condition is FALSE at some stage, otherwise the loop will go on indefinitely.

repeat

```
repeat
{
   some expressions
}
```

In the repeat loop some expressions are repeated ‘infinitely’, so repeat loops will have to contain a break statement to escape them.

See R code for examples

Avoiding Looping : the “apply” functions

- Looping is notoriously slow in R (although it has gotten better over time)
- When working with data frames or matrices (arrays) it is faster to use the various versions of the “apply” functions.

- **lapply** , takes a list input and applies the function given to each element. As its prefix suggests, this function returns a list
- To simplify, we use **sapply** with the s prefix (for *simple*)
- **- tapply** :

```
tapply(X = mtcars$mpg, INDEX = mtcars$cyl, FUN = mean)
4        6        8
26.663 19.743 15.100
```

- - **apply** (to one dimension of an array)
- - **rapply** (recursive)
- - **eapply** (apply to “environments”)

See R code provided

Writing your own function: Syntax of functions

- A function can be regarded as a collection of statements and is an object in R of class `function'.
- Basic syntax:

```
functionname <- function(arg1, arg2,...) {
  Body of function: a collection of valid statements
}
```

- Suppose I want to write a function that takes all possible distances between a set of m locations (the x object) with a set of n locations (the y object) in the 2-d space.
- The result will be an m by n matrix of "cross"-distances between the x and y locations.

```
pairdist = function(x,y)  { m = nrow(x); n = nrow(y)
        for (i in 1:m)  { for (j in 1:n)
                dd = sqrt( (x[i,1]-y[j,1])^2 + (x[i,2]-y[j,2])^2 )
                return(dd)
        }
## if I want to save the function:
save(pairdist,file="pairdist.fsave")
```

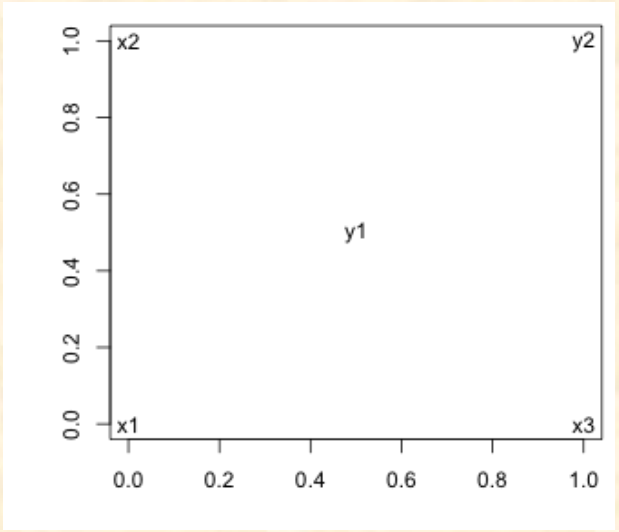
Syntax of functions

- Let's try it

```
x1 = c(0,0); x2 = c(0,1); x3 = c(1,0)
y1 = c(1/2,1/2); y2 = c(1,1)
x = rbind(x1,x2,x3)
y = rbind(y1,y2)
pairdist(x,y)
```

```
> pairdist(x,y)
     [,1]  [,2]
[1,] 0.7071 1.414
[2,] 0.7071 1.000
[3,] 0.7071 1.000
```

Pairwise distance between x1 and y1
Pairwise distance between x2 and y2



Saving and loading Functions

- My own naming convention:
- I save two files:
  makefunction.pairdist.R the script where I compose the function
  pairdist.fsave the R-save file that contains the function (see before)

- Then I can call load the function pairdist later via

```
load("pairdist.fsave")
```

- Alternatively: you can directly "source" the file that contains the function definition:

```
source("makefunction.pairdist.R")
```

- Of course, if you load objects from another directory than your working directory, you will need to provide the entire path, such as in: (on my Mac)

```
source("/Users/localadmin/Documents/1 - Teaching Current/1 - M796 Intro to R/Course Notes
    2012/makefunction.pairdist.R")
```

Required and Optional Arguments

- Arguments inside a function can be made "optional" if a default value is assigned inside the function definition

```
power <- function(x, k=2){
   x^k
   }
power(5)
[1] 25

power()
Error in power() : argument "x" is missing, with no default

power(5,3)
[1] 125
```

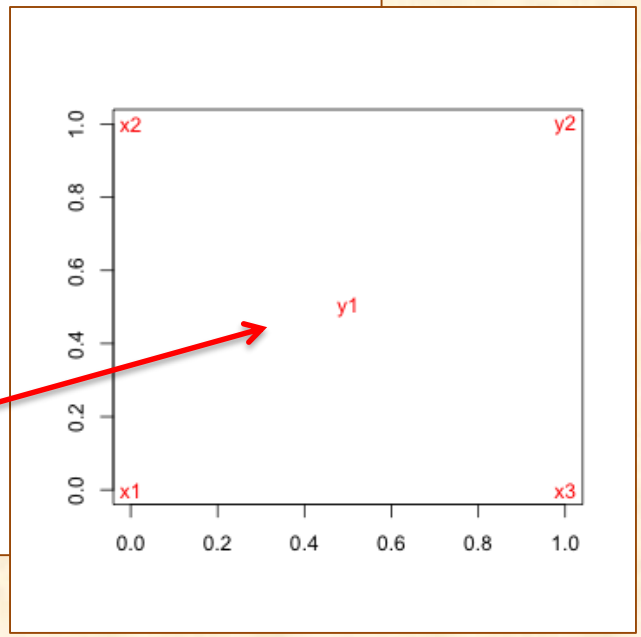
The `...' argument

- The three dots argument can be used to pass arguments from one function to another.
- For example, graphical parameters that are passed to plotting functions or numerical parameters that are passed to numerical routines.

```
pairdistplot = function(x,y,...)
 { m = nrow(x); n = nrow(y)
        dd = matrix(nrow=m,ncol=n)
        for (i in 1:m)  { for (j in 1:n)
          dd[i,j] = sqrt( (x[i,1]-y[j,1])^2 + (x[i,2]-y[j,2])^2 ) }
        print(dd)
        ## make plot
        all = (rbind(x,y))
        plot(all,type="n")  # setup the graph
          text(x,label = paste("x",1:m,sep=""),...)
           text(y,label = paste("y",1:n,sep=""),...)
  }


pairdistplot(a,b, col="red")
```

red colors !



Local and Global Variables

- Assignments of variables inside a function are local, unless you explicitly use a global assignment (the <<- construction or the assign function).
- This means a normal assignment within a function will not overwrite objects outside the function.
- An object created within a function will be lost when the function has finished. Only if the last line of the function definition is an assignment, then the result of that assignment will be returned by the function.

Arguments can be Functions !

- The arguments of a function can be objects of any type, even functions! Consider this:

```
test <- function(n, fun)
        {
        u <- runif(n)
        fun(u)
        }


test(10,sin)
[1] 0.28078332  0.30438298  0.55219120  0.37357375 ...
```

- The second argument of the function test needs to be another function which will be called inside the function.

What does the function return?

- A function usually returns the last expression of its statements
- If you want an earlier expression to be returned, use "print" (see slide 6)
- A function can either return ONE object, or ONE LIST containing MULTIPLE OBJECTS
- To exit a function before it reaches the last line, use the **return** function. Any code after the return statement inside a function will be ignored. For example: (from Lam Guide)

```
myf <- function(x,y){
  z1 <- sin(x)
  z2 <- cos(y)
  if(z1 < 0){
    return( list(z1,z2))
  else{
    return( z1+z2)
  }
}
```

Scoping Rules

- See the Lam Guide for detailed description
- Basically if any object is used inside a function, it will first look among the internal (temporary variables), and then outside among the objects in the working directory and in all the attached libraries, etc.

Lazy evaluation

- See the Lam Guide for detailed description
- This is when a default optional argument uses an input variable
- Note: if you return or print the optional argument, it will evaluate at the current value.

```
myf <- function(x, nc =
length(x))
{
x <- c(x, x)
print(nc)
}
xin <- 1:10
myf(xin)
[1] 20
```

Some References

- **An Introduction to R.** by Longhow Lam ("the Lam Guide")
  https://cran.r-project.org/doc/contrib/Lam-IntroductionToR_LHL.pdf
- Carnegie Mellon U. From a computing class on 'writing R functions':
  http://www.stat.cmu.edu/~cshalizi/402/programming/writing-functions.pdf
- For advanced R programming: Check out Hadley Wickham's website:
  http://adv-r.had.co.nz/
- Institute for Integrative Genome Biology, (UC Riverside) has several online-guides:
  http://manuals.bioinformatics.ucr.edu/
- R for Beginners by Emmanual Paradis. (A short introduction to R in general)
  https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- Vanderbilt Univ. Dept. of Biostatistics: *How to write your own functions and good R programing techniques*
  http://biostat.mc.vanderbilt.edu/wiki/Main/RClinicHelpfulRCode4