R Programming - Part 2

author: Eugene Teo date: February 9th, 2015

Who am i?

https://github.com/eugeneteo/ida-mooc-rprog2

[@eugeneteo](http://www.twitter.com/eugeneteo)

Refresher

“As you learn to program, you are going to get frustrated. You are learning a new language, and it will take time to become fluent. But frustration is not just natural, it’s actually a positive sign that you should watch for.

Frustration is your brain’s way of being lazy; it’s trying to get you to quit and go do something easy or fun. […] If you want to get better at programming, you’ll need to push your brain.

Recognize when you get frustrated and see it as a good thing: you’re now stretching yourself. Push yourself a little further every day, and you’ll soon be a confident programmer."

– Hadley Wickham

Refresher

Coding convention used in these slides

```
(x <- c(1, 2, 3, 4)) # preferred method
```

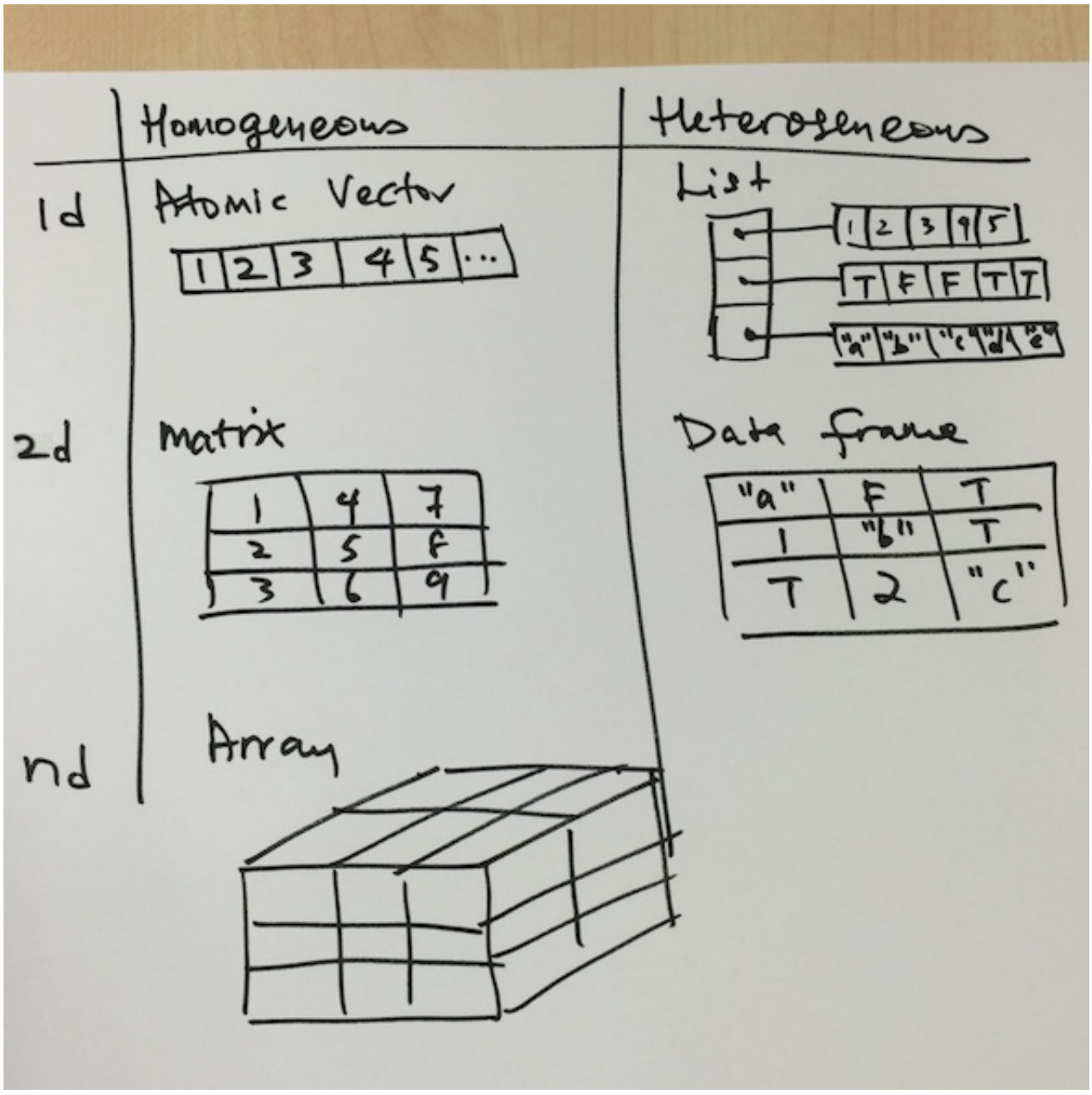
```
[1] 1 2 3 4
```

is the same as

```
x <- c(1, 2, 3, 4)
x
```

```
[1] 1 2 3 4
```

Refresher



Refresher

```
(x <- matrix(1:6, nrow = 2, ncol = 3))
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
seq_len(nrow(x)) # indices of x's rows
```

```
[1] 1 2
```

```
seq_len(ncol(x)) # indices of x's columns
```

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

Refresher

```
(x <- matrix(1:6, nrow = 2, ncol = 3))
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
x[1, 2] # x[row, column]
```

```
[1] 3
```

```
x[2, 3]
```

```
[1] 6
```

Refresher

```
(x <- matrix(1:6, nrow = 2, ncol = 3))
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
for (a in seq_len(nrow(x))) # 1 2
   for (b in seq_len(ncol(x))) # 1 2 3
      print(x[a, b])
```

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

Week 3

Loop functions

Loop Functions

The use of functionals is an alternative to for-loops

- `lapply()` (loops a list and returns a list)
- `sapply()` (simplies the result of `lapply` if possible by returning a vector or matrix or list of values)
- `apply()` (applies a function to margins of an array or matrix, and returns a vector or array or list of values)
- `tapply()` (applies a function to groups of data)
- `mapply()` (applies a function to multiple list or vector arguments)

Loop functions - lapply

Applies a function over a list or vector

```
lapply
```

```
function (X, FUN, ...)
{
    FUN <- match.fun(FUN)
    if (!is.vector(X) || is.object(X))
        X <- as.list(X)
    .Internal(lapply(X, FUN))
}
<bytecode: 0x7fa32102fd78>
<environment: namespace:base>
```

Loop Functions

```
(x <- list(a = 1:3, b = rnorm(3)))
```

```
$a
[1] 1 2 3
```

```
$b
[1] -2.2385283  0.6897759 -0.3757197
```

```
for (a in 1:length(x)) { # 1:2
   print(mean(x[[a]])) # remember [[?]]?
}
```

```
[1] 2
[1] -0.6414907
```

Loop functions - lapply

```
(x <- list(a = 1:3, b = rnorm(3)))
```

```
$a
[1] 1 2 3

$b
[1]  0.4627700 -0.4401947 -1.0321447
```

```
lapply(x, mean) # mean(x$a) and mean(x$b)
```

```
$a
[1] 2

$b
[1] -0.3365231
```

Loop functions - lapply

`runif()` generates random deviates

```
x <- 1:3 # number of observations
lapply(x, runif) # runif(n, min = 0, max = 1)
```

```
[[1]]
[1] 0.4682086

[[2]]
[1] 0.09908585 0.31213501

[[3]]
[1] 0.32114970 0.08100106 0.98748551
```

Loop functions - lapply

```
(x <- list(a = matrix(1:4, nrow=2, ncol=2),
           b = matrix(1:6, nrow=3, ncol=2)))
```

```
$a
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Loop functions - lapply

```
x <- list(a = matrix(1:4, nrow=2, ncol=2),
          b = matrix(1:6, nrow=3, ncol=2))
# show column 1 in matrix format
lapply(x, function(elt) elt[, 1, drop = F])
```

```
$a
     [,1]
[1,]    1
[2,]    2

$b
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

Loop functions - sapply

`sapply()` simplies the result of `lapply()` if possible

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector of the same length (>1), a matrix is returned
- If the result is a list where every element is of different types or lengths, it will silently return a list

Loop functions - sapply

```
data(mtcars) # in-built dataset
str(mtcars[1, ]) # first row
```

```
'data.frame':   1 obs. of  11 variables:
 $ mpg : num 21
 $ cyl : num 6
 $ disp: num 160
 $ hp  : num 110
 $ drat: num 3.9
 $ wt  : num 2.62
 $ qsec: num 16.5
 $ vs  : num 0
 $ am  : num 1
 $ gear: num 4
 $ carb: num 4
```

Loop functions - sapply

If the result is a list where every element is length 1, then a vector is returned

```
data(mtcars) # in-built dataset
# mtcars[1, ] # show first row
sapply(mtcars, is.numeric) # returns a vector
```

```
 mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Loop functions - sapply

If the result is a list where every element is a vector of the same length (>1), a matrix is returned

```
x <- c(8, 8, 8) # 3 columns of 8 observations
sapply(x, runif) # runif(8, min = 0, max = 1)
```

```
          [,1]       [,2]       [,3]
[1,] 0.46275975 0.04335141 0.33915467
[2,] 0.23744824 0.61148808 0.31302731
[3,] 0.73718435 0.08582378 0.51388086
[4,] 0.88424684 0.24862132 0.01103550
[5,] 0.39772136 0.35580143 0.95110089
[6,] 0.04426167 0.58004048 0.02265616
[7,] 0.84149079 0.81558835 0.86771253
[8,] 0.28387279 0.60785803 0.49148457
```

Loop functions - sapply

```
(x <- data.frame(x = 1:10, y = Sys.time() + 1:10)) # plus 1:10 secs
```

```
    x                   y
```

```
1   1 2015-02-07 10:52:04
2   2 2015-02-07 10:52:05
3   3 2015-02-07 10:52:06
4   4 2015-02-07 10:52:07
5   5 2015-02-07 10:52:08
6   6 2015-02-07 10:52:09
7   7 2015-02-07 10:52:10
8   8 2015-02-07 10:52:11
9   9 2015-02-07 10:52:12
10 10 2015-02-07 10:52:13
```

Loop functions - sapply

If the result is a list where every element is of different types or lengths, it will silently return a list

```
x <- data.frame(x = 1:10, y = Sys.time() + 1:10)
sapply(x, class) # returns a list
```

```
$x
[1] "integer"

$y
[1] "POSIXct" "POSIXt"
```

Loop functions - apply

Applies a function to margins of an array or matrix, and returns a vector or array or list of values

```
str(apply)
```

```
function (X, MARGIN, FUN, ...)
```

`MARGIN` is a vector giving the subscripts which the function will be applied over: * `1` indicates rows * `2` indicates columns * `c(1, 2)` indicates rows and columns

Loop functions - apply

```
(x <- matrix(1:16, nrow = 2))
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    3    5    7    9   11   13   15
[2,]    2    4    6    8   10   12   14   16
```

```
apply(x, 1, mean) # row
```

```
[1] 8 9
```

```
apply(x, 2, mean) # col
```

```
[1]  1.5  3.5  5.5  7.5  9.5 11.5 13.5 15.5
```

Loop functions - apply

`quantile()` provides the sample quantiles based on the given probabilities

```
# quantile(x, probs = seq(0, 1, 0.25),
#          na.rm = FALSE, names = TRUE,
#          type = 7, ...)
quantile
```

```
function (x, ...)
UseMethod("quantile")
<bytecode: 0x7fa3231acd88>
<environment: namespace:stats>
```

Loop functions - apply

```
(x <- matrix(rnorm(15), ncol = 3))
```

```
            [,1]       [,2]         [,3]
[1,]  0.20638086 -0.1341630  0.454166012
[2,] -0.31234446  1.7214582  0.992061061
[3,]  1.38489565  1.1261598 -0.568060673
[4,]  0.08508306 -0.8391951  0.009162009
[5,] -0.66024766 -0.2804249  0.759973437
```

```
apply(x, 2, quantile, probs = c(0.25, 0.75))
```

```
          [,1]       [,2]        [,3]
25% -0.3123445 -0.2804249 0.009162009
75%  0.2063809  1.1261598 0.759973437
```

Loop functions - tapply

Applies a function to groups of data using a grouping factor

```
str(tapply)
```

```
function (X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

`X` is a vector and `INDEX` is a grouping factor. The function should expect one argument, which is a vector of elements taken from x according to their group

Loop functions - tapply

```
data(Orange) # Growth of Orange Trees
Orange[1:10, ]
```

```
   Tree  age circumference
1     1  118            30
2     1  484            58
3     1  664            87
4     1 1004           115
5     1 1231           120
6     1 1372           142
7     1 1582           145
8     2  118            33
9     2  484            69
10    2  664           111
```

Loop functions - tapply

```
data(Orange) # Growth of Orange Trees
str(Orange$Tree)
```

```
 Ord.factor w/ 5 levels "3"<"1"<"5"<"2"<..: 2 2 2 2 2 2 2 4 4 4 ...
```

```
levels(Orange$Tree)
```

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

```
nlevels(Orange$Tree)
```

```
[1] 5
```

Loop functions - tapply

```
data(Orange) # Growth of Orange Trees
tapply(Orange$circumference, Orange$Tree, mean) # returns an array
```

```
        3         1         5         2         4
 94.00000  99.57143 111.14286 135.28571 139.28571
```

If `simplify` = `FALSE` (not the default), it will return a list

Loop functions - tapply

```
# 20x standard normals, 10x random deviates
x <- c(rnorm(10), runif(10), rnorm(10, 1))
# generate factor levels
(f <- gl(3, 10)) # 3 = levels, 10 = replications
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

```
tapply(x, f, mean)
```

```
         1          2          3
-0.3662969  0.4416382  0.5412807
```

Loop functions - tapply

Returns a list instead of an array, see `simplify = FALSE`

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
tapply(x, f, mean, simplify = FALSE)
```

```
$`1`
[1] 0.09139896

$`2`
[1] 0.5737328

$`3`
[1] 1.411114
```

Loop functions - tapply

`range()` returns a vector of min and max

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
# returns an array with the mode of the scalar
tapply(x, f, range)
```

```
$`1`
[1] -3.033554  1.792158

$`2`
[1] 0.05541106 0.97794027

$`3`
[1] -1.777545  2.537002
```

Loop functions - mapply

Applies a function to multiple list or vector arguments

```
str(mapply)
```

```
function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

Loop functions - mapply

```
mapply(rep, 1:3, 3:1)
```

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

[[2]]
[1] 2 2

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

```
# rep(1, 3) [1], 2, 3  [3], 2, 1
# rep(2, 2) 1, [2], 3  3, [2], 1
# rep(3, 1) 1, 2, [3]  3, 2, [1]
```

Loop functions - mapply

```
l1 <- list(a = c(1:10), b = c(11:20))
l2 <- list(c = c(21:30), d = c(31:40))
# sum the corresponding elements of l1 and l2
mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
 [1]  64  68  72  76  80  84  88  92  96 100
```

```
# sum(1, 11, 21, 31) = 64
```

```
# sum(2, 12, 22, 32) = 68
# sum(3, 13, 23, 33) = 72
# ...
# sum(8, 18, 28, 38) = 92
# sum(9, 19, 29, 39) = 96
# sum(10, 20, 30, 40) = 100
```

Loop functions - split

Takes a vector or other objects and splits it into groups determined by a factor or list of factors

```
str(split)
```

```
function (x, f, drop = FALSE, ...)
```

`x` is a vector or data frame containing values to be dividend into groups

`f` is a factor (or coerced to one) or a list of factors

`drop` indicates if empty factor levels should be dropped. FALSE by default

Loop functions - split

```
x <- c(rnorm(5), runif(5), rnorm(5, 1))
f <- gl(3, 5) # 3 levels, 5 replications
split(x, f) # returns a list of vectors
```

```
$`1`
[1] -1.3841932 -0.1466192 -1.4490820  0.2533132 -0.2254770

$`2`
[1] 0.8376879 0.1411234 0.5043843 0.7396503 0.9647902

$`3`
[1] 1.3078085 1.7957987 0.2445747 2.5139787 0.6559782
```

Loop functions - split

```
x <- c(rnorm(5), runif(5), rnorm(5, 1))
f <- gl(3, 5) # 3 levels, 5 replications
lapply(split(x, f), mean)
```

```
$`1`
[1] -0.009444776

$`2`
[1] 0.5304554
```

```
$`3`
[1] 0.6045341
```

Loop functions - split

```
data(airquality)
# first three months
s <- split(airquality, airquality$Month)[1:3]
# column mean
sapply(s, function(x) apply(x[, c("Ozone", "Solar.R", "Wind", "Temp")], 2, mean,
na.rm = TRUE))
```

```
                5         6          7
Ozone    23.61538  29.44444  59.115385
Solar.R 181.29630 190.16667 216.483871
Wind     11.62258  10.26667   8.941935
Temp     65.54839  79.10000  83.903226
```

Loop functions - split

```
x <- 1:10
(f1 <- gl(2, 5)) # 2 - levels, 5 - repl
```

```
 [1] 1 1 1 1 1 2 2 2 2 2
Levels: 1 2
```

```
f2 <- gl(5, 2) # 5 - levels, 2 - replications
interaction(f1, f2) # Factor interactions
```

```
 [1] 1.1 1.1 1.2 1.2 1.3 2.3 2.4 2.4 2.5 2.5
Levels: 1.1 2.1 1.2 2.2 1.3 2.3 1.4 2.4 1.5 2.5
```

Loop functions - split

```
x <- 1:10
f1 <- gl(2, 5) # 2 - levels, 5 - repl
f2 <- gl(5, 2)
str(split(x, list(f1, f2)))
```

```
List of 10
 $ 1.1: int [1:2] 1 2
 $ 2.1: int(0)
 $ 1.2: int [1:2] 3 4
```

```
$ 2.2: int(0)
$ 1.3: int 5
$ 2.3: int 6
$ 1.4: int(0)
$ 2.4: int [1:2] 7 8
$ 1.5: int(0)
$ 2.5: int [1:2] 9 10
```

Loop functions - split

Explanation:

```
# f1 = 11 11 1 2 22 22
# f2 = 11 22 3 3 44 55
# x  = 12 34 5 6 78 910
#       2  2 1 1  2  2
# There's 1.1, 1.2, 1.3, 2.3, 2.4, and 2.5
# There's no 2.1, 2.2, 1.4, and 1.5
```

Read Splitting on more than one level

Not covering

Debugging tools and R profiling

References

R Programming by Roger D. Peng, Jeff Leek and Brian Caffo

Advanced R by Hadley Wickham

R Cookbook by Paul Teetor

A brief introduction to “apply” in R by Neil Saunders

R tapply Function by endmemo.com

Thanks

Join the first and second cohorts’ Facebook groups!

[@eugeneteo](http://www.twitter.com/eugeneteo)