# Loop and vectorized programming in R

Hunyong Cho
Department of Biostatistics

# Loop and vectorized programming in R

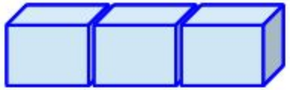
| | |
|---|---|
| loop | for / while / repeat |
| *apply | apply / lapply / sapply / tapply |
| others | Reduce / map / do.call / ... |

# Preliminaries

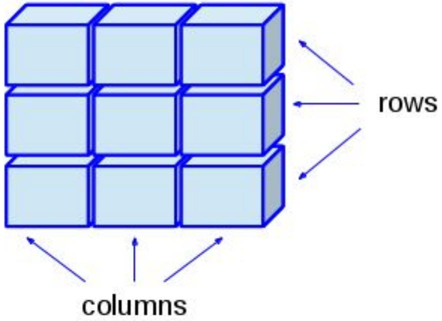
Basic data structures in R

- vector
- matrix
- array
- list
- data.frame

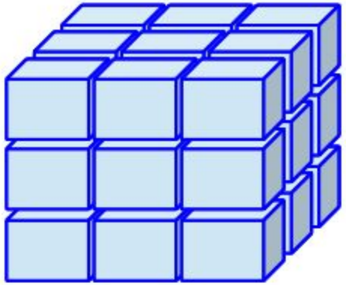
Vector



Matrix

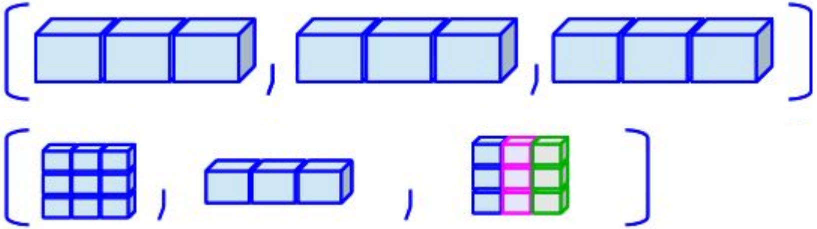


Array

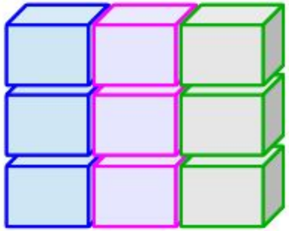


List

[ 1,  “A”,  c(“a”, “b”, “c”) ]



Data Frame
(Table)



source: https://devopedia.org/r-data-structures

# Basic data structures in R

- vector
- matrix
- array

Homogeneous

- list
- data.frame

Heterogeneous

How to get access to the elements?

Tell R the desired address

- by numbers
- by names

with the following operators

- [ ]    for vector, matrix, array
- [ ], [[ ]], or $    for list, data.frame

Let’s download the data.

```
abc <- read.table("exercise.dat", header=T)
```

# Loop

# Motivating example

**Question**:
Want to draw histogram for each variable.
Is there an easy way?

```
hist(abc$y1)
hist(abc$y2)
...
hist(abc$y7)
```

# Avoid manual repetition by using loops

**answer: loop**

```
for (i in 3:9) {
  hist(abc[, i])
}
```

**Some cosmetics**

```
for (i in 3:9) {
  hist(abc[, i], main = names(abc)[i])
}
```

# for loop

**Syntax**

```
for (i in <range of loop>) {
   <things to do>
}
```

**Example**

```
for (i in 1:10) {
   print(i^2)
}
```

# for loop

**Example 2**

Write a code to calculate $1^2 + 2^2 + ... + 10^2$.

```
a <- 0

for (i in 1:10) {
   a <- a + i^2
}

print(a)
```

# for loop

**Tip: The range does not have to be numeric.**

```
for (i in names(abc)) {
   print(i)
}
```

# repeat loop

```
repeat {
   <things to do>
   <stopping rule>
}
```

**Example**

```
i = 1
repeat {
   print(i^2)
   if (i >100) { break }
}
```

# while loop

```
while (condition) {
   <things to do>
}
```

**Example**

```
i = 1
while (i <= 100) {
   print(i^2)
   i = i + 1
}
```

# Exercise

**Can you replace this code using names in the for loop range?**

```
for (i in 3:9) {
  hist(abc[, i])
}
```

# vectorized operations

# Motivating example

**This is a very inefficient way of coding for**

$$1^2 + 2^2 + \dots + 10^2.$$

```
a <- 0

for (i in 1:10) {
    a <- a + i^2
}

print(a)
```

But you could have simply done:

```
sum( (1:10)^2 )
```

# Exercise

1. Code the following:

   log(2) * log(3) * ... * log(10)

2. Get the average of the following:

   expit(6), expit(7), ... , expit(10)

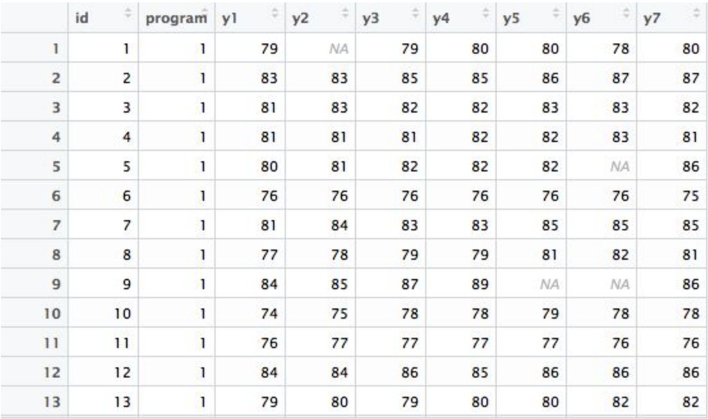
   Note expit(x) = exp(x) / {exp(x) + 1}.
   In R, you can use `plogis(x)`

# vectorized operations:

`apply`

# Motivating example

How can you get the sum of each column of `abc`?



| | id | program | y1 | y2 | y3 | y4 | y5 | y6 | y7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 79 | NA | 79 | 80 | 80 | 78 | 80 |
| 2 | 2 | 1 | 83 | 83 | 85 | 85 | 86 | 87 | 87 |
| 3 | 3 | 1 | 81 | 83 | 82 | 82 | 83 | 83 | 82 |
| 4 | 4 | 1 | 81 | 81 | 81 | 82 | 82 | 83 | 81 |
| 5 | 5 | 1 | 80 | 81 | 82 | 82 | 82 | NA | 86 |
| 6 | 6 | 1 | 76 | 76 | 76 | 76 | 76 | 76 | 75 |
| 7 | 7 | 1 | 81 | 84 | 83 | 83 | 85 | 85 | 85 |
| 8 | 8 | 1 | 77 | 78 | 79 | 79 | 81 | 82 | 81 |
| 9 | 9 | 1 | 84 | 85 | 87 | 89 | NA | NA | 86 |
| 10 | 10 | 1 | 74 | 75 | 78 | 78 | 79 | 78 | 78 |
| 11 | 11 | 1 | 76 | 77 | 77 | 77 | 77 | 76 | 76 |
| 12 | 12 | 1 | 84 | 84 | 86 | 85 | 86 | 86 | 86 |
| 13 | 13 | 1 | 79 | 80 | 79 | 80 | 80 | 82 | 82 |

2977 2925 2931 2955 2784 2454 2454

1. for loop

```
for (i in 3:9) {
    sum(abc[, i], na.rm = T) %>% print
}
```

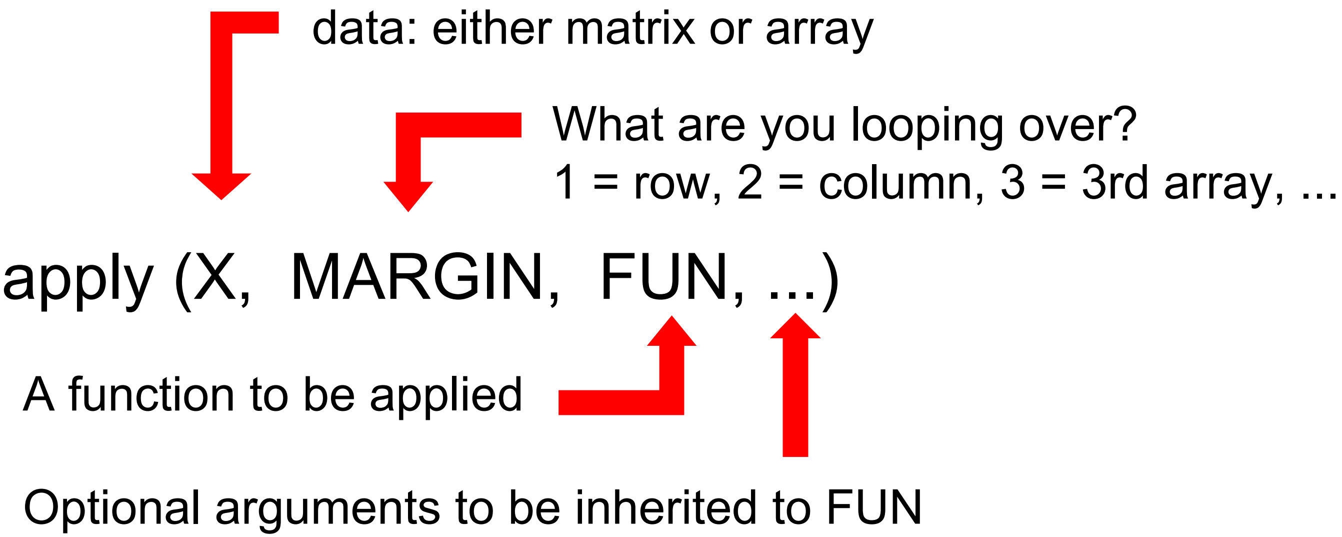
2. apply

2 means columns, 1 means rows

```
apply (abc, 2, sum)

apply (abc, 2, sum, na.rm = T)
```

# Syntax of apply



data: either matrix or array

What are you looping over?
1 = row, 2 = column, 3 = 3rd array, ...

apply (X,  MARGIN,  FUN, ...)

A function to be applied

Optional arguments to be inherited to FUN

# Exercise

1. For each observation of abc, get the mean of y1 to y7 (using apply).
   (hint: Use the subset of the data. abc[, 3:9])

2. For each observation of abc, get the trimmed mean of y1 to y7 (using apply).

   (hint: mean(..., trim = 0.2))

# Exercise, continued

3. From the following data array,
   get a dataset averaged across centers:

```
data.by.center = array(1:27, c(3, 3, 3),
                       dimnames = list(sample = 1:3,
                                       variable = c("y1", "y2", "y3"),
                                       center = LETTERS[1:3]))
```

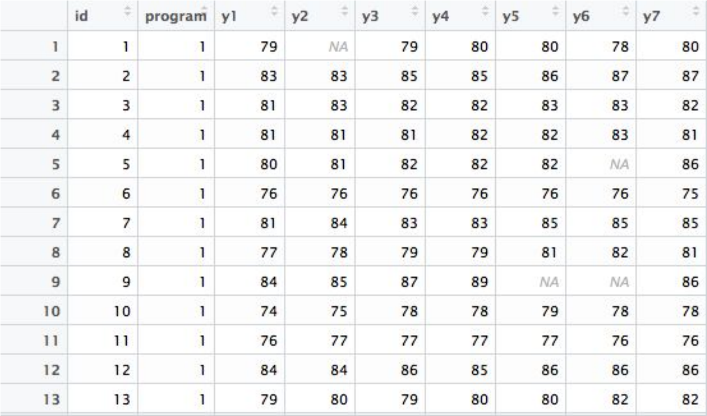
```
apply(data.by.center, c(1,2), mean)
```

# vectorized operations:

lapply

# Motivating example

How can you store a list of tables?
Suppose we want to save a table for each variable (y1 to y7) in abc.



| | id | program | y1 | y2 | y3 | y4 | y5 | y6 | y7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 79 | NA | 79 | 80 | 80 | 78 | 80 |
| 2 | 2 | 1 | 83 | 83 | 85 | 85 | 86 | 87 | 87 |
| 3 | 3 | 1 | 81 | 83 | 82 | 82 | 83 | 83 | 82 |
| 4 | 4 | 1 | 81 | 81 | 81 | 82 | 82 | 83 | 81 |
| 5 | 5 | 1 | 80 | 81 | 82 | 82 | 82 | NA | 86 |
| 6 | 6 | 1 | 76 | 76 | 76 | 76 | 76 | 76 | 75 |
| 7 | 7 | 1 | 81 | 84 | 83 | 83 | 85 | 85 | 85 |
| 8 | 8 | 1 | 77 | 78 | 79 | 79 | 81 | 82 | 81 |
| 9 | 9 | 1 | 84 | 85 | 87 | 89 | NA | NA | 86 |
| 10 | 10 | 1 | 74 | 75 | 78 | 78 | 79 | 78 | 78 |
| 11 | 11 | 1 | 76 | 77 | 77 | 77 | 77 | 76 | 76 |
| 12 | 12 | 1 | 84 | 84 | 86 | 85 | 86 | 86 | 86 |
| 13 | 13 | 1 | 79 | 80 | 79 | 80 | 80 | 82 | 82 |

```
table(abc$y1)
table(abc$y2)
...
```

# Motivating example

Suppose we want to save a density plot for each variable (y1 to y7) in abc.

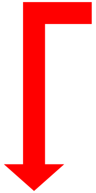
1. for loop

```
result <- list()
for (i in 1:9) {
  result[[i]] <- table(abc[, i])
}
```

2. lapply

```
lapply (abc, table)
```

# Syntax of lapply



data: a list (including data.frame)

lapply (X,  FUN, ...)

# Exercise 1

Update the following list by removing redundant values (using lapply).

```
set.seed(1)
xyz <-
    list(fruit = c("apple", "banana", "apple", "grape", "tomato"),
        letters = sample(letters, 15),
        numbers = sample(1:10, 15, replace = TRUE))
```

```
lapply (xyz, unique)
```

# Exercise 2

Get the number of distinct values for each element of xyz (using lapply).

From this code, lapply (xyz, unique)

solution 1)

```
xyz %>% lapply(unique)
      %>% lapply(length)
```

solution 2)

```
lapply(xyz, function(x) length(unique(x)))
```

# vectorized operations:

`sapply`

# Motivating example

Consider the following code:

```
lapply (xyz, length)
```

The output is again a list. Instead of a list, a vector is enough.

How can we simplify the result?

solution 1)

```
lapply(xyz, length) %>% unlist
```

solution 2)

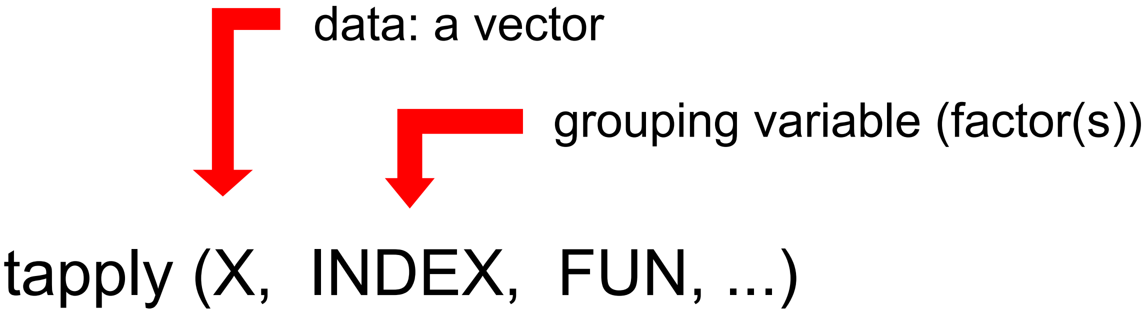
```
sapply(xyz, length)
```

# vectorized operations:

## tapply

(similar to group_by + summarize)

# Syntax of tapply



data: a vector

grouping variable (factor(s))

tapply (X,  INDEX,  FUN, ...)

# Exercise

For each program in abc, get the average of y1.

```
tapply (abc$y1, abc$program, mean)
```

# vectorized operations:

`map`

# map as an alternative to lapply

```
lapply (abc, table)

purrr::map (abc, table)
```

# vectorized operations:

## Reduce

# Motivating example

1 + 2 + 3 + 4 + 5 + ... + 100

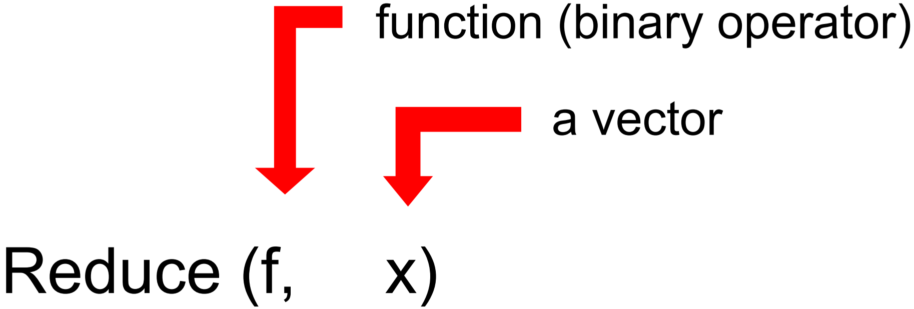
→ `sum(1:100)`

If we do not have `sum` function, how can we do this simply?

solution

```
Reduce("+", 1:100)
```

# Syntax of Reduce



function (binary operator)

a vector

Reduce (f, x)

≡ ... f(f(f(x1, x2), x3), x4)...

# Exercise

Calculate the following using Reduce

Define an operator x ++ y = 2x + y
and calculate the following: 1 ++ 2 ++ 3 ++ ... ++ 10
or ... 2* (2* (2* 1 + 2) + 3) + 4 ...

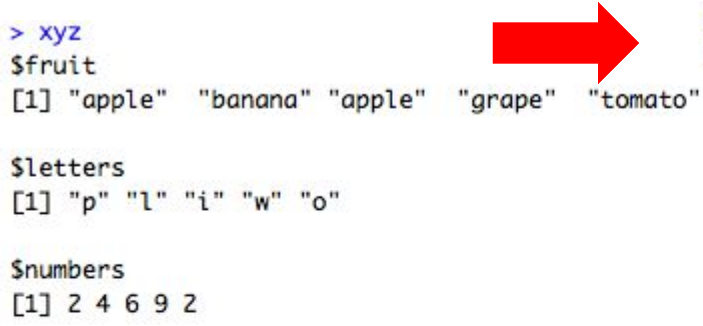
```
Reduce (function(x, y) 2*x + y, 1:10)
```

# vectorized operations:

do.call

# Motivating example

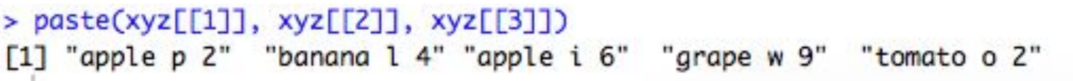
Want to put elements of a list into a function each as an argument.



```
> xyz
$fruit
[1] "apple"  "banana" "apple"  "grape"  "tomato"

$letters
[1] "p" "l" "i" "w" "o"

$numbers
[1] 2 4 6 9 2
```



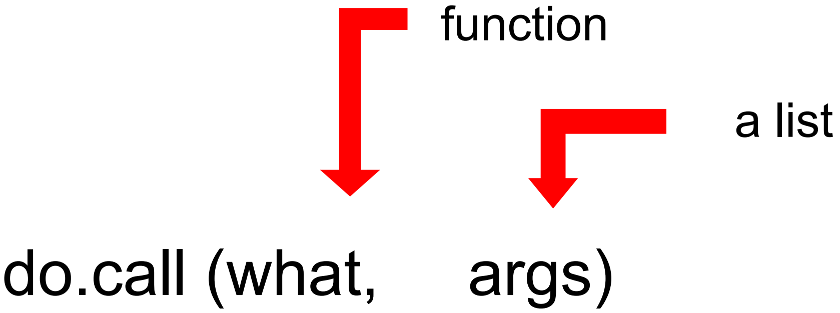
```
> paste(xyz[[1]], xyz[[2]], xyz[[3]])
[1] "apple p 2"  "banana l 4" "apple i 6"  "grape w 9"  "tomato o 2"
```

How can we do this simply?

solution

do.call(paste, xyz)

# Syntax of do.call



function

a list

do.call (what, args)

≡ what ( args[[1]], args[[2]], ..., args[[n]] )

# Exercise

Make a matrix by column-wise combining (i.e cbind) the list elements without repetition but using do.call.

```
cbind(xyz[[1]], xyz[[2]], xyz[[3]])
```

```
do.call (cbind, xyz)
```

vectorized operations:

other useful functions

# Other useful vector-related functions

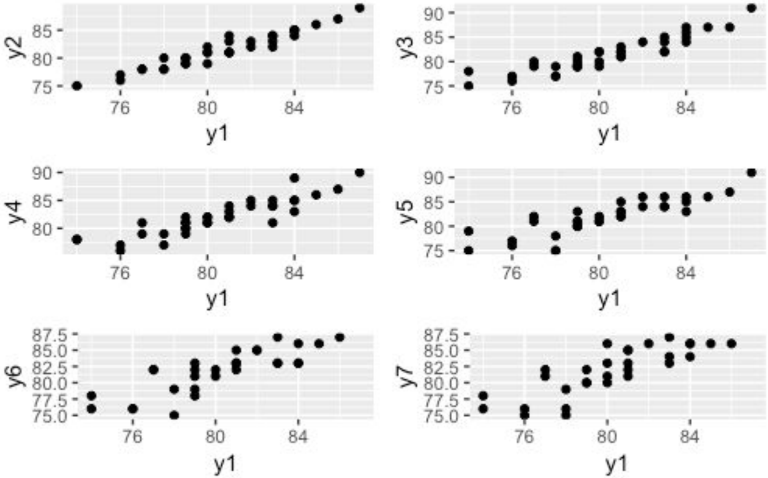
| | |
|---|---|
| expand.grid | e.g. expand.grid(LETTERS, 1:3) |
| outer | e.g. outer(1:3, 1:3, "+") |
| Vectorize | |
| | |
| ifelse | |
| which / which.min / which.max | |
| pmin / pmax | |
| cumsum | |
| cumprod | |
| | |
| vector | e.g. vector("list", 10) |

# Exercise

# Exercise

Make an arranged list of `qplot`s (plotting y2, ..., y7 against y1) using

`lapply` , `do.call`, and `gridExtra::grid.arrange`



hint: `qplot(abc[, 3], abc[, i], xlab = "y1", ylab = names(abc)[i])`
`grid.arrange(ggplot1, ggplot2, ..., ggplot6)`