Introduction to R[*]

Will Heo and ChienHsun Lin

8/25/2023

1 Package

A package is a collection of R functions, data, and code in a well-defined format. Many users have contributed by making convenient and powerful functions that make coding enjoyable. Some packages provide functions that operate much faster, and other packages provide functions that simplify tedious jobs such as data cleaning and analysis. There are more than 10,000 packages, but it is important to know some main packages that are commonly used by many.

Packages are stored in the directories in R, which are called libraries. To use functions in a package, you need to do the following two steps:

```
install.packages("ggplot2")
library(ggplot2)
```

You need to install the package once. Once it is installed, you simply need to load the package by running the second line. Make sure to include the quotation marks when installing a package.

2 Vector and Matrix

Every vector has two key properties, type and length.There are various types of vectors: logical, integer, double, charater, complex, raw and list. Integer and double vectors are called numeric vectors.

```
typeof(letters)
```

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

```
typeof(1:10)
```

```
## [1] "integer"
```

```
x <- list("a", "b", 1:10) #list can contain another lists.
length(x)
```

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

[*]This note is drawn from many sources such as "R for Data Science" by Hadley Wickham and Garrett Grolemund and "DataCamp".

```r
# Logical
1:10 %% 3 == 0 #divided by 3
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
```

For numeric vectors, integers have NA while doubles have four: NA, NaN, Inf and -Inf. NaN, Inf and -Inf can arise during division:

```r
c(-1, 1, 0) / 0
```

```
## [1] -Inf  Inf  NaN
```

You can coerce one type of vector to another. Explicit coercion happens when using as.logical(), as.integer(), as.double() or as.character(). Implicit coercion example:

```r
x <- sample(20, 100, replace = TRUE)
y <- x > 10 #TRUE is converted to 1 and FALSE is converted to 0
sum(y)
```

```
## [1] 39
```

```r
mean(y)
```

```
## [1] 0.39
```

2.1 Test functions: is_type()

```r
is_integer(x)
```

```
## [1] TRUE
```

```r
is_integer(y)
```

```
## [1] FALSE
```

```r
is_logical(y)
```

```
## [1] TRUE
```

2.2 Naming vectors

```r
c(x = 1, y = 2, z = 4)
```

```
## x y z
## 1 2 4
```

```r
set_names(1:3, c("a", "b", "c"))
```

```
## a b c
## 1 2 3
```

2.3 Subsetting

```r
x <- c("one", "two", "three", "four", "five")
x[c(3, 2, 5)]
```

```
## [1] "three" "two"   "five"
```

```r
x <- c(abc = 1, def = 2, xyz = 5)
x[c("xyz", "def")]
```

```
## xyz def
##   5   2
```

2.4 Matrix

The syntax for matrix: matrix(value, nrow, ncol, byrow, dimnames) byrow is TRUE or FALSE. If TRUE, the elements of the matrix are arranged in the row, whereas FALSE will arrange the element by column-wise.

```r
mat = matrix(1:10, nrow = 2, ncol = 5, byrow = TRUE)
print(mat)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```r
mat = matrix(1:10, nrow = 5, ncol = 2, byrow = F)
print(mat)
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```r
mat <-
  matrix(1:10, nrow = 2, dimnames = list(c("r1","r2"), c("c1","c2","c3","c4","c5")))

mat
```

```
##    c1 c2 c3 c4 c5
## r1  1  3  5  7  9
## r2  2  4  6  8 10
```

```
mat[2,3]
```

```
## [1] 6
```

2.5 Multiplication of Matrix

```
mat1<- matrix(c(2,4,6,8), nrow = 2, ncol = 2)
print(mat1)
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

```
mat2 <- matrix(c(14,16,18,20), nrow = 2, ncol = 2)
print(mat2)
```

```
##      [,1] [,2]
## [1,]   14   18
## [2,]   16   20
```

```
mat1 * mat2
```

```
##      [,1] [,2]
## [1,]   28  108
## [2,]   64  160
```

```
mat1 %*% mat2
```

```
##      [,1] [,2]
## [1,]  124  156
## [2,]  184  232
```

3 If..else

```
# A simple example
a <- 10
b <- 200

if (b > a) {
  print("b is greater than a")
}
```

```
## [1] "b is greater than a"
```

```r
#using else if
a <- 10
b <- 10

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print ("a and b are equal")
}
```

```
## [1] "a and b are equal"
```

```r
if (b > a) {
  print("b is greater than a")
} else if (a < b) {
  print("b is smaller than a")
} else {
  print("a and b are equal")
}
```

```
## [1] "a and b are equal"
```

```r
#ifelse
ifelse(b == a, "a and b are equal", "a and b are different")
```

```
## [1] "a and b are equal"
```

4 Loops

Just as other programming languages, you can declare loops to do repetitive works for you. There are two main classes of loop environments: `for` and `while`.

4.1 `for`

`for` executes a set of commands for certain times.

```r
nums <- numeric(15)
# create a vector with 15 numeric elements (default at 0).

for (x in 1:10) {
  nums[x] = x - 1
  #make the xth element of nums x-1
}

nums
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9 0 0 0 0 0
```

```r
for (y in 1:length(nums)) {
  nums[y] = 15 - y
}

nums
```

```
##  [1] 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

```r
num_matrix <- matrix(nums, nrow = 3, ncol = 5)

#Create transpose matrix with nested for loops
trans_matrix <- matrix(numeric(15), nrow = 5, ncol = 3)

for (x in 1:ncol(num_matrix)) {
  for (y in 1:nrow(num_matrix)) {
    trans_matrix[x,y] <- num_matrix[y,x]
  }
}

trans_matrix
```

```
##      [,1] [,2] [,3]
## [1,]   14   13   12
## [2,]   11   10    9
## [3,]    8    7    6
## [4,]    5    4    3
## [5,]    2    1    0
```

```r
for (x in 1:length(num_matrix)) {
  old_v = num_matrix[x]
  if (mod(num_matrix[x],5) == 0) {
    num_matrix[x] = 100
  } else {
    next #break and escape from the loop
  }
  print(paste("replaced the element ", old_v))
}
```

```
## [1] "replaced the element  10"
## [1] "replaced the element  5"
## [1] "replaced the element  0"
```

```r
num_matrix
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   14   11    8  100    2
## [2,]   13  100    7    4    1
## [3,]   12    9    6    3  100
```

```r
for (x in 1:length(num_matrix)) {
  if (num_matrix[x] >= 7) {
    num_matrix[x] = sqrt(num_matrix[x] - 7)
  } else {
    print("I cannot conduct this manipulation: there exists an element in negative domain.")
    break #break and escape from the loop
  }
}
```

```
## [1] "I cannot conduct this manipulation: there exists an element in negative domain."
```

```r
num_matrix
```

```
##          [,1]     [,2] [,3] [,4] [,5]
## [1,] 2.645751 2.000000    1  100    2
## [2,] 2.449490 9.643651    0    4    1
## [3,] 2.236068 1.414214    6    3  100
```

4.2 while

while is a loop that continues to loop over when a condition is met (true). This means that a while loop would run forever if there is no variable that triggers the condition to be false inside the loop.

```r
# Syntax: while ([condition]) { [command here] }

truth_v <- TRUE
while_num <- 1

while (truth_v) {
  while_num <- while_num + 1
  truth_v <- (while_num < 4)
  print(c(while_num, truth_v))
}
```

```
## [1] 2 1
## [1] 3 1
## [1] 4 0
```

```r
# Equivalently,

while_num <- 1

while (while_num < 4){
  while_num <- while_num + 1
  truth_v <- (while_num < 4)
  print(c(while_num, truth_v))
}
```

```
## [1] 2 1
## [1] 3 1
## [1] 4 0
```

You can also use `next` and `break` for `while`.

5 apply Functions

Because of the fundamental data structure of R, it is more efficient to run the for loops with apply function.

```
# make your matrix a data frame
## Note: You can still use matrix as the input of apply
data<-as.data.frame(num_matrix)

data
```

```
##         V1       V2 V3  V4  V5
## 1 2.645751 2.000000  1 100   2
## 2 2.449490 9.643651  0   4   1
## 3 2.236068 1.414214  6   3 100
```

```
apply(data, MARGIN = 1, FUN = sum)
```

```
## [1] 107.64575  17.09314 112.65028
```

```
apply(data, MARGIN = 2, FUN = sum)
```

```
##         V1         V2         V3         V4         V5
##   7.331309  13.057864   7.000000 107.000000 103.000000
```

```
sum <- c()
for (i in 1:ncol(data)) {
  sum <- c(sum, sum(data[,i]))
}
sum
```

```
## [1]   7.331309  13.057864   7.000000 107.000000 103.000000
```

6 Pipes (Ctrl + Shift + M)

It comes from **magrittr** package and **tidyverse** also loads it automatically. It is used to chain a sequence of calculations.

```
x <- runif(10) #uniform distribution.
#You can generate random number using also rnorm, rbinom, rexp, sample(interger)
#Without pipe
round(exp(diff(log(x))),1)
```

```
## [1]  0.9  7.3  0.2  9.1  0.1 10.2  0.5  0.1 12.9
```

```
#With pipe
x %>% log() %>%
  diff() %>%
  exp() %>%
  round(1)
```

```
## [1]  0.9  7.3  0.2  9.1  0.1 10.2  0.5  0.1 12.9
```

```
library(babynames)
data("babynames")
head(babynames)
```

```
## # A tibble: 6 x 5
##    year sex   name          n   prop
##   <dbl> <chr> <chr>     <int>  <dbl>
## 1  1880 F     Mary       7065 0.0724
## 2  1880 F     Anna       2604 0.0267
## 3  1880 F     Emma       2003 0.0205
## 4  1880 F     Elizabeth  1939 0.0199
## 5  1880 F     Minnie     1746 0.0179
## 6  1880 F     Margaret   1578 0.0162
```

```
# Without a pipe
sum(select(filter(babynames, sex == "M", name == "Taylor"), n))
```

```
## [1] 109852
```

```
# With pipes
babynames %>% filter(sex == "M", name == "Taylor") %>%
  select(n) %>%
  sum
```

```
## [1] 109852
```

7 Functions

You can define your own functions.

```
# Basic syntax: function(argument1, argument2 = [default value]) { [command here] }

transpose_mat <- function(MAT) {
  for (x in 1:ncol(MAT)) {
    for (y in 1:nrow(MAT)) {
      trans_matrix[x,y] <- MAT[y,x]
    }
  }
  return(trans_matrix) #use return() to specify the output of the function.
}

transpose_mat(num_matrix)
```

```
##            [,1]     [,2]       [,3]
## [1,]   2.645751 2.449490   2.236068
## [2,]   2.000000 9.643651   1.414214
## [3,]   1.000000 0.000000   6.000000
## [4,] 100.000000 4.000000   3.000000
## [5,]   2.000000 1.000000 100.000000
```

```r
min_col_mat <- function(MAT, col = 1) {
  min_n <- min(MAT[,col])
  return(min_n)
}

min_col_mat(num_matrix)
```

```
## [1] 2.236068
```

```r
min_col_mat(num_matrix, col = 2)
```

```
## [1] 1.414214
```

```r
min_col_mat(num_matrix, 3)
```

```
## [1] 0
```

```r
min_max_mat <- function(MAT) {
  max_x <- c()
  for (x in 1:ncol(MAT)) {
      max_x <- c(max_x, max(MAT[,x]))
  }
  min_max <- min(max_x)
  return(min_max)
}

min_max_mat(num_matrix)
```

```
## [1] 2.645751
```

```r
min_max_mat(transpose_mat(num_matrix))
```

```
## [1] 9.643651
```

8 R Markdown

To produce a complete report containing all text, code, and results, click “Knit” or press Cmd/Ctrl + Shift + K.

8.1 Headers

A single hashtag creates a first level header and as the number of hashtags increases, the size of the header decreases.

8.2 Italic and Bold text

Two stars in each side are used for bold texts and one star in each side is used for Italic text. **This is bold** and *This is italic*

8.3 List

Each bullet point begins with one asterisk. Leave a blank line before the first bullet.

- First bullet
  - First sub-bullet
    - sub-sub bullet
- Second bullet

To make a numbered list, use 1 instead of asterisks.

1. First numbered item
2. Second numbered item

8.4 Line breaks

Line ends with
two or more spaces for the end of line.

8.5 Links

Use a plain http address or add a link to a phrase.
http://www.githup.com
Github

8.6 Mathmatical formula

Use dollar signs as in LaTeX.

$$y = x + 10$$

8.7 Showing codes and results in the document

- Default: shows codes and results at the same time

```
mat1
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

- Hiding results: add the argument eval = FALSE inside the brackets and after r.

```
mat1
```

- Hiding codes: add the argument echo = FALSE inside the brackets and after r.

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```