Post1

Sangwook Kim

2017 10 23

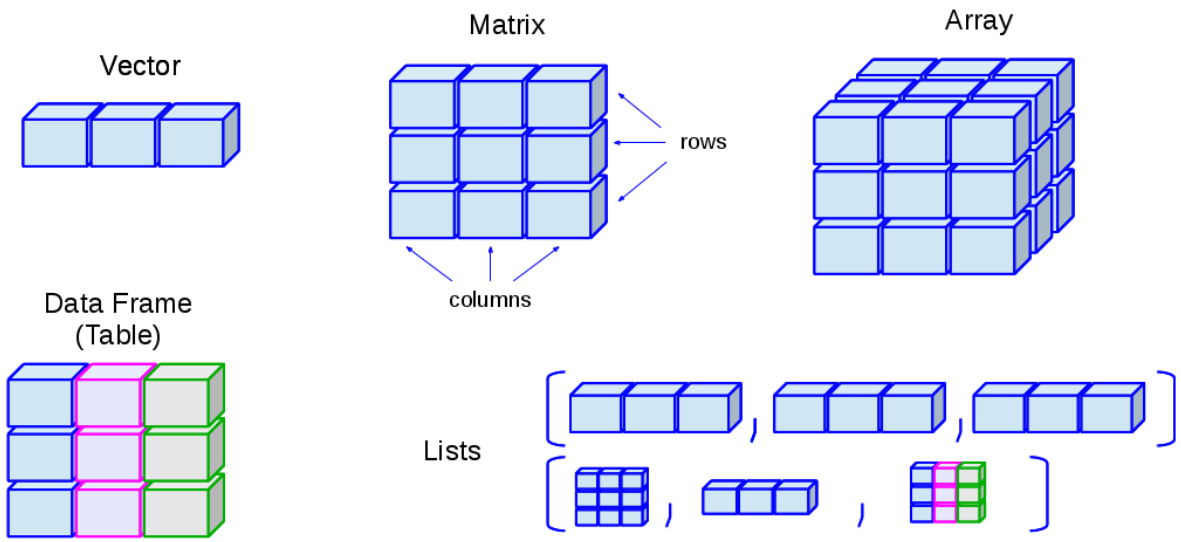
List

Introduction

The purpose of this post is to explore the concept of list in R and to introduec interesting examples of list so that it gives you deep and precise understanding of list in R.

For R based data structures, we can categorize them into two aspects: one's dimension and whether one contains all same type contents(homogeneous) or not
(heterogenous).

List is 1-dimensional heterogenous data structure. Vecture is 1-dimensional data structure but homogeneous and Data frame is heterogenous but 2-dimensional data structure.



We can create a list with list():

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

Since list is heterogenous data structure unlike atomic vector, it can contains different types of elements in it.

```
x <- list('1', 2L, 3.4, TRUE)

str(x)
```

```
## List of 4
##  $ : chr "1"
##  $ : int 2
##  $ : num 3.4
##  $ : logi TRUE
```

Also, it can contain list itself and 1-dimensional data structure such as vector.

```
y <- list(1, 2, 3)
x <- list(y, c("a", "b", "c"))

x
```

```
## [[1]]
## [[1]][[1]]
## [1] 1
##
## [[1]][[2]]
## [1] 2
##
## [[1]][[3]]
## [1] 3
##
##
## [[2]]
## [1] "a" "b" "c"
```

You can name list elements and access each elements by list$names

```r
list_name <- list(a=c(1:4), b="b", c=c("a","b","c"))

names(list_name) # It gives a vector of names in list.
```

```
## [1] "a" "b" "c"
```

```r
list_name$a <- c(4:1) # Access each elements by its names and maniplutate it.

names(list_name) <- c("A", "B", "C") # Assign name for each elements.
```

List manipulation: Indexing and Editing.
Let's see how to manipulate each elements in list.

```r
# Creat a list
m1 <- list(first=c(1:4), "b", third=c("a","b","c"))

# By using '$', we can look at the elements inside of list.

m1$first # access by name
```

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

```r
m1[2] # access by indexing since there is no name for 2nd elements in the list m1
```

```
## [[1]]
## [1] "b"
```

Let's change the value of first and second elements in m1

```r
# We can do like this
m1$first <- c(5:10)
m1[2] <-  "c"

# Check m1
m1
```

```
## $first
## [1]  5  6  7  8  9 10
##
## [[2]]
## [1] "c"
##
## $third
## [1] "a" "b" "c"
```

Yes, the first and second elements of m1 have been changed !

Here's an interesting fact about list. Unlike vector, list does not have the concept of vectorization.

```r
# Create an vector
a <- c(1, 2, 3)

# Create a list
b <- list(1, 2, 3)
```

```r
# Possible
a <-  a * 2

# Impossible to complie it
b <-  b * 2
```

Let's parse structure of a list

```r
b <- list("a", 1, TRUE)

# It is "list"
typeof(b)
```

```
## [1] "list"
```

```r
# What about this?
typeof(b[1])
```

```
## [1] "list"
```

```
# How avout this?
typeof(b[[1]])
```

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

```
typeof(b[[2]])
```

```
## [1] "double"
```

```
typeof(b[[3]])
```

```
## [1] "logical"
```

```
# What is length?
length(b)
```

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

It seems that list is a tree structure in which each elements are distinct lists containing atomic vector or types.

How to apply the concept of vectorization to list?

It is not quite simple because list in R can have various types and also need to break a list structure into small atomic value.

Let's start from a simple list

```
l <- list(1, 2, 3)
```

I want to add 1 to each elements in the list 'l'

```
l[[1]] <- l[[1]] + 1
l[[2]] <- l[[2]] + 1
l[[3]] <- l[[3]] + 1
```

It works fine but not quite burdensome. So let's use function and loop to generalize what we've done above

```
list_add_one <- function(list){
  for(i in c(1:length(list))){
    list[[i]] <-  list[[i]] + 1
  }
  list
}

t <- list(4, 5, 6)
list_add_one(t)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 7
```

But list can contains list inside of it, then how to manipulate it? Then we need to think of type of elements and use condition.

```
p <- list(1,2, list(3,4))
typeof(p[[1]])
```

```
## [1] "double"
```

```
typeof(p[[3]])
```

```
## [1] "list"
```

```
list_add_one <- function(list){
  for(i in c(1:length(list))){
    if(typeof(list[[i]]) == "list"){
      # if the type of element is list then call the function for it.
      list[[i]] <- list_add_one(list[[i]]) # it is recursive step for "list" type element.
    }
    else{
    list[[i]] <-  list[[i]] + 1
    }
  }
  list
}

# It works for nested list.
list_add_one(p)
```

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

```
q <- list(c(1,2,3), list(list(4,5), 6))

# Also works..
list_add_one(q)
```

```
## [[1]]
## [1] 2 3 4
##
## [[2]]
## [[2]][[1]]
## [[2]][[1]][[1]]
## [1] 5
##
## [[2]][[1]][[2]]
## [1] 6
##
##
## [[2]][[2]]
## [1] 7
```

Useful packages for manipulating List

First, let's explore **purrr** pacakage(https://cran.r-project.org/web/packages/purrr/index.html).
In this package, we are going to focus on two useful fucntions to manipluate list:**lapply()**, **sapply()**

Let's call **purrr** library

```
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 3.4.2
```

Description of **lapply** (https://www.rdocumentation.org/packages/base/versions/3.4.1/topics/lapply)

The syntax is lapply(list or vector, function)

For example:

```
l1 <- list(x=c(1,2,3), y=c(3,4,5), z=c(100,0,10))

lapply(l1, mean)
```

```
## $x
## [1] 2
##
## $y
## [1] 4
##
## $z
## [1] 36.66667
```

lapply() gives the mean of each elements in list form.

Then what if we want a vector as a result? Let's use sapply():

```
sapply(l1, mean)
```

```
##        x        y        z
##  2.00000  4.00000 36.66667
```

Here is the description of **sapply** (https://www.rdocumentation.org/packages/memisc/versions/0.99.14.3/topics/Sapply)

So we don't need to worry about the concept of vectorization in list by making an complicated function. we rather use powerful built-in-function, lapply and saaply.

What else we can do with lapply() and sapply()?

Let's look at this code

```
list1 <- lapply(1:2, function(x) matrix(NA, nrow = 2, ncol = 2))
class(list1)
```

```
## [1] "list"
```

We made a list having by using lapply !

How about this one ?

```
m <- sapply(1:2, function(x) matrix(NA, nrow = 2, ncol = 2))
class(m)
```

```
## [1] "matrix"
```

This is a matrix form.

What is the practical uses of lists in r ?

Also, list is the basic form of data frame
Look at this code:

```
dat <- list(col1=(c(1:3)),
            col2=(c(4:6)),
            col3=(c(7:9)))
dat <- as.data.frame(dat)
```

We can make data frame by using list.

Another very useful package to manipulate List

rlist is a very useful package when you want to do list mapping, filtering, grouping, sorting thing on list like a dplyr on vector.

```
library(rlist)
```

This is the manual for rlist.

Let's say we mainly using list on our whatever data we have.
For example, we have a list like this:

```
dat <-
  list(
    p1=list(name="Jason",age=24,
      interest=c("reading","music","guitar"),
      language=list(r=2,csharp=4)),
    p2=list(name="Jamine",age=23,
      interest=c("sports","music", "dancing"),
      language=list(r=3,java=2,cpp=5)),
    p3=list(name="Jack",age=22,
      interest=c("sports","driving", "reading"),
      language=list(r=1,cpp=4,python=2)))
```

Like what we have learned in 'dplyr', we can do such thing on list by using functions in rlist.

Let's filter who like reading and has been using R for more than 2 years
We can do it by using 'list.filter()' like this:

```
list.filter(dat, "reading" %in% interest & language$r >= 2)
```

```
## $p1
## $p1$name
## [1] "Jason"
##
## $p1$age
## [1] 24
##
## $p1$interest
## [1] "reading" "music"   "guitar"
##
## $p1$language
## $p1$language$r
## [1] 2
##
## $p1$language$csharp
## [1] 4
```

Let's select one's age and interest.
We can do it by using 'list.select()' like this:

```
list.select(dat, age, interest)
```

```
## $p1
## $p1$age
## [1] 24
##
## $p1$interest
## [1] "reading" "music"   "guitar"
##
##
## $p2
## $p2$age
## [1] 23
##
## $p2$interest
## [1] "sports"  "music"   "dancing"
##
##
## $p3
## $p3$age
## [1] 22
##
## $p3$interest
## [1] "sports"  "driving" "reading"
```

Let's map each one to the number of one's language capability.
We can do it by using 'list.map()' like this:

```
list.map(dat, length(language))
```

```
## $p1
## [1] 2
##
## $p2
## [1] 3
##
## $p3
## [1] 3
```

Some miscellaneous fact about list in R

What is Pairlist ?

When you carefully look at the description of list, you would see 'pairlist'.
You might not give any attention but let's see what pairlist is.

First of all, pairlist is not important in normal use in R because list mostly do all the works but pairlist is mostly used in terms of R intervals

Although we're not studying R intervals, but let's just see the difference between list and pairlist. The main difference is the way that each list is stored. Pairlist is stored as a chain of nodes, in which each node is linked to the next node's location(address). On the other hand, in list, each contents is stored in one location. To see more details