Note 5. List

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

List Objects

- List can combine objects of different types (different modes & different data types).
- A list object is a vector (recursive vector).
- Creating list objects:
  - `list()`: By typing components or combining existing objects.
  - `vector(mode='list',length)`: Empty storage, initialized list.

```
> x <- list(name = 'Kim',salary=6000,union=T); x
$name
[1] "Kim"

$salary
[1] 6000

$union
[1] TRUE
```

Creating Lists

```
> x$name
[1] "Kim"
> x <- list('Kim',6000,T); x
[[1]]
[1] "Kim"
[[2]]
[1] 6000
[[3]]
[1] TRUE

> x[[1]]
[1] "Kim"

> a <- 1:3; b <- c('a','b','c')
> x <- list(a,b); x
[[1]]
[1] 1 2 3
[[2]]
[1] "a" "b" "c"
```

Creating Lists

```
> x <- list(N=a,K=b)
> x
$N
[1] 1 2 3

$K
[1] "a" "b" "c"

> x <- vector(mode='list',2)
> x
[[1]]
NULL

[[2]]
NULL
```

List Indexing

- Extracting components from list objects:
  (1) `list$component`: Names of components can be abbreviated to whatever extent is possible without causing ambiguity.
  (2) `list[[`*index*`]]`.
  (3) `list[['component']]`

- Single bracket `[ ]` can also access list components. But, the result is another list.

```
> x <- list(name = 'Kim',salary=6000,union=T)
> x$salary
[1] 6000
> x$sal
[1] 6000
> x[[2]]
[1] 6000
> x[['salary']]
[1] 6000
```

List Indexing

```
> x[2]
$salary
[1] 6000

> x[1:2]
$name
[1] "Kim"

$salary
[1] 6000

> class(x[2])
[1] "list"
>
> x[[1:2]]
Error in x[[1:2]] : subscript out of bounds
> class(x[[2]])
[1] "numeric"
```

Adding or Deleting List Components

- List components can be added by creating or concatenating them.
- List components can be deleted by setting them to NULL.

```
> x <- list(a=1:2,b=c('a','b'))
> x$c <- rep(3,5)
> x
$a
[1] 1 2
$b
[1] "a" "b"
$c
[1] 3 3 3 3 3

> y <- list(1:2); c(y,list(c('a','b')))
[[1]]
[1] 1 2
[[2]]
[1] "a" "b"
```

Adding or Deleting List Components

```
> x[[4]] <- c(T,F,F,T)
> x[5:6] <- c(7,3)
> x
$a
[1] 1 2
$b
[1] "a" "b"
$c
[1] 3 3 3 3 3
[[4]]
[1]  TRUE FALSE FALSE  TRUE
[[5]]
[1] 7
[[6]]
[1] 3
```

Adding or Deleting List Components

```
> x$c <- NULL; x
$a
[1] 1 2
$b
[1] "a" "b"
[[3]]
[1]  TRUE FALSE FALSE  TRUE
[[4]]
[1] 7
[[5]]
[1] 3

> x[3:5] <- NULL; x
$a
[1] 1 2
$b
[1] "a" "b"
```

LIST OPERATIONS

- # of components in a list: `length()` ($\because$ list is vector).
- `names()`: Names of list components.
- `unlist()`: Accessing values in list components.

```
> x <- list(a=1:2,b=c('a','b')); length(x)
[1] 2

> names(x)
[1] "a" "b"

> y <- unlist(x); y
 a1  a2  b1  b2
"1" "2" "a" "b"
> is.vector(y)
[1] TRUE
```

List Operations

```
> x <- list(a=1:5,b=10:15)
> y <- unlist(x)
> y
a1 a2 a3 a4 a5 b1 b2 b3 b4 b5 b6
 1  2  3  4  5 10 11 12 13 14 15

> class(y)
[1] "integer"

> names(y) <- NULL
> y
 [1]  1  2  3  4  5 10 11 12 13 14 15
```

Applying Functions to Lists

- `lapply(`*`list, function`*`)`: Apply *function* to each component of *list* $\Rightarrow$ Result: list object.
- `sapply(`*`list, function`*`)` $\Rightarrow$ Result: vector or matrix object.

```
> x <- list(a = 1:10, b=c(5,8,1,7))
> y <- lapply(x,median); y
$a
[1] 5.5
$b
[1] 6

> y <- sapply(x,median); y
  a   b
5.5 6.0
```

Applying Functions to Lists

```
> sapply(x,sort)
$a
 [1]  1  2  3  4  5  6  7  8  9 10
$b
[1] 1 5 7 8

> lapply(x,sort)
$a
 [1]  1  2  3  4  5  6  7  8  9 10
$b
[1] 1 5 7 8

> x <- list(a=1:5,b=5:1)
> sapply(x,sort)
     a b
[1,] 1 1
[2,] 2 2
[3,] 3 3
[4,] 4 4
[5,] 5 5
```

Recursive Lists

- Recursive list: List within lists.

```
> x <- list(a=1:2, b=c('a','b','c'))
> y <- list(x=c('k','j')); z = list(x,y); z
[[1]]
[[1]]$a
[1] 1 2
[[1]]$b
[1] "a" "b" "c"
[[2]]
[[2]]$x
[1] "k" "j"

> z[[1]]
$a
[1] 1 2
$b
[1] "a" "b" "c"

> z[[1]]$b
[1] "a" "b" "c"
```