

Introduction to R*

Lecture 4: Heterogeneous vectors (Lists & Dataframes) and IO

Wim R.M. Cardoen

Last updated: 10/19/2022 @ 12:34:03

Contents

1	R Lists	2
1.1	Creation of a list	2
1.1.1	Examples	2
1.2	Accessor operators [], [[]], \$ in R.	5
1.2.1	General statements	5
1.2.2	Homogenous vectors	5
1.2.2.1	Examples	5
1.2.3	Heterogeneous vectors (i.e. lists and derived classes)	6
1.2.3.1	Examples	6
1.3	Modifying lists	8
1.3.1	Examples	8
1.4	Functions: return multiple objects	8
1.4.1	Example	8
1.5	Exercises	10
2	R Dataframes	11
2.1	Examples	11
2.2	Creating a data frame	11
2.3	attach and detach	11
3	Input-Output (IO)	12
3.1	Functionality in Base R	12
3.2	Other options:	12

*© - Wim R.M. Cardoen, 2022 - The content can neither be copied nor distributed without the **explicit** permission of the author.

In the first part of this section, two kinds¹ of **heterogeneous** vectors will be discussed:

- lists
- data frames (& tibbles)

Input-output (IO) in R forms the subject of the latter part.

1 R Lists

A **list** is a heterogeneous vector that **may** contain one or more **components**.

The components can be **heterogeneous** objects (atomic types, functions, lists², ...).

Under the hood, the list is implemented as a vector of pointers to its top-level components.

The list's length equals the number of top-level components.

1.1 Creation of a list

An R list can be created in several ways:

- using the `list()` function (most common)
- via the `vector()` function
- via a cast using the `as.list()` function

1.1.1 Examples

- use of the `list()` function

```
# Creating an empty list
x1 <- list()
x1
```

```
list()
```

```
cat(sprintf("  typeof(x1):%s  class(x1):%s  length(x1):%d\n",
            typeof(x1), class(x1), length(x1)))
```

```
typeof(x1):list  class(x1):list  length(x1):0
```

```
# A more 'complicated' version
x2 <- list(1:10, c("hello", "world"),
          3+4i, matrix(data=1:6, nrow=2, ncol=3, byrow=TRUE))
x2
```

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

¹R also has the pairlist. This topic will not be discussed in this section. People interested in this subject, should have a look at [R-internals](#).

²Due to this feature, they are also called **recursive** vectors.

```
[[2]]
[1] "hello" "world"
```

```
[[3]]
[1] 3+4i
```

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

```
cat(sprintf("  typeof(x2):%s  class(x2):%s  length(x2):%d\n",
            typeof(x2), class(x2), length(x2)))
```

```
typeof(x2):list  class(x2):list  length(x2):4
```

```
# Using existing names
```

```
x3 <- list(x=1, y=2, str1="hello", str2="world", vec=1:5)
x3
```

```
$x
[1] 1
```

```
$y
[1] 2
```

```
$str1
[1] "hello"
```

```
$str2
[1] "world"
```

```
$vec
[1] 1 2 3 4 5
```

```
# Applying name to list
```

```
x4 <- list(matrix(data=1:4,nrow=2,ncol=2), c(T,F,T,T), "hello")
names(x4) <- c("mymat","mybool","mystr")
x4
```

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

```
$mybool
[1] TRUE FALSE TRUE TRUE
```

```
$mystr  
[1] "hello"
```

- use `vector()` function:

Allows to create/allocate an empty vector of a certain length.

```
# Allocate a vector of length 5  
x5 <- vector(mode="list", length=5)  
x5
```

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

```
[[2]]  
NULL
```

```
[[3]]  
NULL
```

```
[[4]]  
NULL
```

```
[[5]]  
NULL
```

- using the `as.list()` function

```
x6 <- as.list(matrix(5:10,nrow=2))  
x6
```

```
[[1]]  
[1] 5
```

```
[[2]]  
[1] 6
```

```
[[3]]  
[1] 7
```

```
[[4]]  
[1] 8
```

```
[[5]]  
[1] 9
```

```
[[6]]  
[1] 10
```

Note: The ‘inverse’ operation is `unlist()`

```
x7 <- unlist(x6)  
x7
```

```
[1] 5 6 7 8 9 10
```

1.2 Accessor operators [], [[]], \$ in R.

1.2.1 General statements

The operator `[[i]]` selects **only one component** (in cases of lists) or **only one element** in case of homogeneous vectors.

The operator `[]` allows to select **one or more components** (in the case of lists) or **one or more elements** in the case of homogeneous vectors.

The `$` operator can **only** be used for **generic/recursive vectors**.

If you use the `$` operator to other objects you will obtain an **error**.

The `$` operator can **only** be followed by a string or a non-computable index.

1.2.2 Homogenous vectors

In praxi, for homogeneous vectors there is not much difference between `[[]]` and `[]` **except** that `[[]]` does **NOT** allow to select more than **one** element.

Note: The operator `[[]]` can be used as a tool of defensive programming.

1.2.2.1 Examples

```
a <- seq(from=1,to=30,by=3)
a
```

```
[1] 1 4 7 10 13 16 19 22 25 28
```

```
# Extraction of ONE element
cat(sprintf(" a[[2]] : %d\n", a[[2]]))
```

```
a[[2]] : 4
```

```
cat(sprintf(" a[2] : %d\n", a[2]))
```

```
a[2] : 4
```

```
# Extraction of MORE than 1 element using [[ ]] => ERROR
a[[c(2,3)]]
```

```
Error in a[[c(2, 3)]]: attempt to select more than one element in vectorIndex
```

but:

```
# Extraction of MORE than 1 element using [ ] => OK
a[c(2,3)]
```

```
[1] 4 7
```

1.2.3 Heterogeneous vectors (i.e. lists and derived classes)

We stated earlier that the operator `[[]]` allows to select **only one** component.

It also means that this operator selects **the component as is**.

If this one component is a matrix it selects a matrix, if it were a list it will be a list, etc.

The operator `[]` allows to select more than **one** component.

Therefore, in order to return potentially heterogeneous components it **always** returns a **list** even if only one component were to be returned.

1.2.3.1 Examples

```
x2
```

```
[[1]]
```

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

```
[[2]]
```

```
[1] "hello" "world"
```

```
[[3]]
```

```
[1] 3+4i
```

```
[[4]]
```

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

```
# Selection using [[]]
```

```
x24 <- x2[[4]]
```

```
x24
```

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

```
class(x24)
```

```
[1] "matrix" "array"
```

```
typeof(x24)
```

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

```
length(x24)
```

```
[1] 6
```

```
# Selection using []
```

```
x24 <- x2[4]
```

```
x24
```

```
[[1]]
```

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

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

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

```
class(x24)
```

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

```
typeof(x24)
```

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

```
length(x24)
```

```
[1] 1
```

```
# Select third el. of the FIRST component
```

```
x13 <- x2[[1]][3]
```

```
x13
```

```
[1] 3
```

which is the same as:

```
v1 <- x2[[1]]
```

```
v1[3]
```

```
[1] 3
```

Heterogeneous vectors are also known as **recursive/generic** vectors, as can be seen in the following example:

```
# A more advanced 'recursive' example
```

```
v <- list(v1=1:4,  
          lst1=list(a=3, b=2, c=list(x=5,y=7, v2=seq(from=7,to=12))))
```

```
# Extracting the component as a homogeneous vector
```

```
v[[2]][[3]][[3]]
```

```
[1] 7 8 9 10 11 12
```

```
class(v[[2]][[3]][[3]])
```

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

```
# Extracting as a list  
v[[2]][[3]][3]
```

```
$v2  
[1] 7 8 9 10 11 12
```

```
class(v[[2]][[3]][3])
```

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

We can extract the same data using names if available:

```
v$lst1$c$v2
```

```
[1] 7 8 9 10 11 12
```

```
# List of function objects  
lstfunc <- list(cube=function(x){x**3},  
               quartic=function(x){x**4})  
lstfunc$cube(5)
```

```
[1] 125
```

```
lstfunc$quartic(5)
```

```
[1] 625
```

1.3 Modifying lists

- modifying elements
- inserting elements
- deleting elements
- concatenating lists

1.3.1 Examples

1.4 Functions: return multiple objects

If a function needs to return **multiple objects** a list **must** be used.

1.4.1 Example

```
func01 <- function(n)  
{  
  x <- n*(n+1)/2  
  y <- cbind(1:n, (1:n)^2)
```



```
    return(list('x'=x, 'y'=y))
}
n <- 8
res <- func01(n)
```

```
res$x
```

```
[1] 36
```

```
res$y
```

	[,1]	[,2]
[1,]	1	1
[2,]	2	4
[3,]	3	9
[4,]	4	16
[5,]	5	25
[6,]	6	36
[7,]	7	49
[8,]	8	64

1.5 Exercises

- Let's consider the following list:

```
lstex1 <- list(
  list(
    seq(from=4,to=40,by=5),
    "hello world",
    list(
      matrix(100:119,nrow=5),
      "bye",
      c(7,13,17),
      list(451,-1,-17)
    )
  )
)
```

Extract the following data from mylistex1:

- the elements 7 13 as a vector.
 - the second column of `matrix(100:119,nrow=5)` as a matrix.
 - -1 as a scalar.
 - -1 as a list.
 - **all** numerical values into a vector. (Hint:`unlist()`)
- Create the following list:

```
lstex2 <- list(1:10,
  matrix(seq(from=1,to=20), nrow=4),
  5+3i,
  list('a','d','b'),
  "UoU",
  c(T,F,T,T,T),
  "Hello"
)
```

Perform some operations (deletions, insertions, modifications) such that `lstex2` takes on the following form:

```
lstex2 <- list( matrix(seq(from=1,to=20), nrow=4),
  "Hello UoU",
  list('a','b','c','d'),
  c(T,F,T,T,T)
)
```

2 R Dataframes

A `data frame` is a `list` with three `attributes`:

- `names` : component names
- `row.names` : row names
- `class`: `data.frame`

From the above we can infer that the number of rows is the **same** for each component. The components of a data frame are either vectors, factors, numerical matrices, lists or other data frames.

2.1 Examples

2.2 Creating a data frame

- `read.table`
- `data.frame`

2.3 attach and detach

3 Input-Output (IO)

3.1 Functionality in Base R

3.2 Other options:

- library `readr`
 - supports a lot of formats (csv, tcsv, delim, ...)
 - allows column specification
 - faster than Base R's read/write operations
 - uses a `tibble` instead of a `data frame`.
 - for more info: [R for Data Science - Chapter 11.Data import](#)
- library `data.table`
 - very fast IO: optimal for large read (`fread()`) and write (`fwrite()`) operations
 - memory efficient
 - low-level parallelism (use of multiple CPU threads)