Introduction to R*

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

Wim R.M. Cardoen

Last updated: 10/17/2022 @ 20:21:18

Contents

In the first part of this section, two kinds[1] 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, lists[2], ...).

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

[1] R 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.
[2] Due 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 Accessing elements

1.2.1 Examples

1.3 Modifying lists

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

1.3.1 Examples

1.4 Notes

1.4.1 Adding a return a list

1.4.2 More on [[]], vs. []

1.5 Exercises

2 R Dataframes

A `date 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)