Introduction to R

R Markdown

R is an object-oriented language: everything in R is an object.

R has 6 basic data types:

- character

  : "a", "ciao"

- numeric

  : 2, 15.5

- integer

  : 2L (the L tells R to store this as an integer)

- logical

  : TRUE, FALSE (aka boolean)

- complex

  : 1+4i (complex numbers with real and imaginary parts)

Atomic means that the vector only holds data of a single data type.

Examining objects

R provides many functions to examine features of vectors and objects in general, eg:

- class() - what kind of object is it (high-level)?
- typeof() - what is the object's data type (low-level)?
- length() - how long is it? What about two dimensional objects?
- attributes() - does it have any metadata?

Example

```
x <- "Hola"
typeof(x)
```

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

```
attributes(x)
```

```
## NULL
```

```
s <- 1:5
s
```

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

```
length(s)
```

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

```r
typeof(s)
```

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

```r
#convert to numeric
num_s <- as.numeric(s)
num_s
```

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

```r
typeof(num_s)
```

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

Data structures

R has many data structures. The main ones are the following:

- atomic vector
- list
- matrix
- data frame
- factors
- time series

Vectors

The most common and basic data structure in R. Vectors can be one of two types:

- atomic vectors
- lists

In general the term *vector* is used in reference to the atomic types only.

A vector is a collection of elements usually of mode: character, logical, integer or numeric. it is a one-dimensional data structure and it is homogeneous, i.e. each element is of the same type.

Create a vector

To create an empty vector use vector(). The default mode is logical. You can be more explicit (as shown in the examples below.) It is more common to use direct constructors such as character(), numeric(), etc.

```r
vector()
```

```
## logical(0)
```

```r
# a vector of mode 'character' composed of 5 elements
vector("character", length = 5)
```

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

```r
# same same, but using only the constructor directly.
character(5) # a 'character' vector composed of 5 elements
```

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

```r
numeric(5)   # a 'numeric' vector with 5 elements
```

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

One can create a vector directly specifying their content. R will guess the appropriate mode of storage.

```
# this will be treated as double precision real numbers
v <- c(1, 2, 3)
# to force treating as integer numbers
v_num <- c(1L, 2L, 3L)
# to force treating as integer numbers it can be casted as well
v <- as.integer(c(1, 2, 3))

#create a vector of mode logical
y <- c(TRUE, TRUE, FALSE, FALSE)

#create a vectore of mode character
z <- c("Fabi", "Oriol", "Alessandro", "Jasper", "Alex", "David", "Alejandro", "Laura", "Jasper", "Kevin
```

Examining vectors

```
typeof(v)
```

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

```
length(v)
```

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

```
class(v)
```

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

```
str(v)
```

```
##  int [1:3] 1 2 3
```

```
typeof(z)
```

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

```
length(z)
```

```
## [1] 10
```

```
class(z)
```

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

```
str(z)
```

```
##  chr [1:10] "Fabi" "Oriol" "Alessandro" "Jasper" "Alex" "David" "Alejandro" ...
```

Adding elements to a vector

```
#add elements one by one
en_z <- c(z, "Kevin")
en_z <- c(z, "Ariadna")
en_z
```

```
##  [1] "Fabi"       "Oriol"      "Alessandro" "Jasper"     "Alex"
##  [6] "David"      "Alejandro"  "Laura"      "Jasper"     "Kevin"
## [11] "Ariadna"
```

```r
#add two elements at the same time basically concatenating vectors
En_z <- c(z, c("Kevin", "Ariadna"))
En_z
```

```
##  [1] "Fabi"       "Oriol"      "Alessandro" "Jasper"     "Alex"
##  [6] "David"      "Alejandro"  "Laura"      "Jasper"     "Kevin"
## [11] "Kevin"      "Ariadna"
```

Create Vectors from Sequences

```r
#sequence of numbers
int_series <- 1:5
int_series
```

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

```r
series <- 1.5:5.5
series
```

```
## [1] 1.5 2.5 3.5 4.5 5.5
```

```r
#same same
seq(10)
```

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

```r
seq(from = 1, to = 5, by = 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Selecting elements

```r
#sequence of numbers
int_series <- 1:5
int_series[c(1:2,5)]
```

```
## [1] 1 2 5
```

Special values

NA = Not Available

```r
x <- c(0.5, NA, 0.7)
#check if there are NA elements one by one
is.na(x)
```

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

```r
#check if there is any NA element
anyNA(x)
```

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

Inf = infinite

```r
4/0
```

```
## [1] Inf
```

NaN = Not a Number = undefined value.

```r
0/0
```

```
## [1] NaN
```

```r
Inf/Inf
```

```
## [1] NaN
```

```r
Inf/0
```

```
## [1] Inf
```

Vectors with mixed types elements

```r
#r automatically converts to a single type
m1 <- c(1.7, "a")
m1
```

```
## [1] "1.7" "a"
```

```r
m2 <- c(TRUE, 2)
m2
```

```
## [1] 1 2
```

```r
m3 <- c(FALSE, 3)
m3
```

```
## [1] 0 3
```

```r
m4 <- c("a", TRUE)
m4
```

```
## [1] "a"    "TRUE"
```

Matrices

Multi dimensional atomic vectors, as such they are homogeneous structures containing just one data type. Generally 2 dimensions: rows and columns.

Create a matrix

```r
m <- matrix(nrow = 3, ncol = 4)
m
```

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

```r
#print dimensions nrows and ncols
dim(m)
```

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

```r
#high level type
class(m)
```

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

```r
#low level type
typeof(m)
```

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

```
m <- matrix(c(1,3))
m
```

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

```
#print dimensions nrows and ncols
dim(m)
```

```
## [1] 2 1
```

```
#high level type
class(m)
```

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

```
#low level type
typeof(m)
```

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

```
mq <- matrix(
  c(1, 1, 2, 3),
  nrow = 2,
  ncol = 6)
mq
```

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

Matrices are filled columns wise.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```

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

Select matrix elements

To reference a matrix element: specify the index along each dimension in single square brackets, e.g. m[row_index, column_index].

```
mq[1,4]
```

```
## [1] 2
```

```
mq[2,4]
```

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

```
typeof(mq[1,1]) == typeof(mq)
```

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

```
mq[9] #this selects the ninth element column-wise
```

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

Create a matrix from a vector

```
v <- 1:10
#assigning dimension to the matrix
dim(v) <- c(2, 5)
#dim(v) <- c(5, 2)
v
```

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

Create a matrix binding rows or columns

- rbind() to bind rows
- cbind() to bind columns

NB Dimesions must match!

```
c1 <- 3:6
c2 <- 5:8
cbind(c1, c2)
```

```
##      c1 c2
## [1,]  3  5
## [2,]  4  6
## [3,]  5  7
## [4,]  6  8
```

```
r1 <- 3:6
r2 <- 5:8
r3 <- 9:12
rbind(c1, c2, r3)
```

```
##    [,1] [,2] [,3] [,4]
## c1    3    4    5    6
## c2    5    6    7    8
## r3    9   10   11   12
```

It's good to know that the following expressions result in NaNs and infinite.

```
0/0
```

```
## [1] NaN
```

```
Inf/Inf
```

```
## [1] NaN
```

```
Inf/0
```

```
## [1] Inf
```

List

R lists are versatile containers since, unlike atomic vectors, they can have mixed data types. Aka generic vectors, lists are (in general) heterogeneous data containers which can also have another list as element (nested list).

A list is a special type of vector. Each element can be a different type.

- To create a list: list()
- To convert objects to list: as.list().

An empty list of the required length can be created using vector()

Create a list

```
x <- list(5, "hola", TRUE, 1.618)
x
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] "hola"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1.618
```

Select list elements

A list's element can be referenced using double square brackets. NB index starts from 1.

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

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

Since a list is a special type of vector, it can be created also with vector().

```
#create an empty list
x <- vector("list", length = 4)
length(x)
```

```
## [1] 4
```

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

```
## NULL
```

Convert vector to list.

```
x <- 1:5
x <- as.list(x)
length(x)
```

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

Lists: elements names

Elements of a list can be named (i.e. lists can have the names attribute)

```
l<- list(a = "Ciao", b = 1:5, data = head(iris))
names(l)
```

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

```
l$a
```

```
## [1] "Ciao"
```

```
l[["b"]]
```

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

```
#Using a single bracket another list is returned label+element
l[1]
```

```
## $a
## [1] "Ciao"
```

Functions in R returns only one object, so lists can be handy since they are versatile containers that can have labels for elements.

On the console each element of a printed list starts on a new line.

Dataframes

It is maybe the most popular data structure, perfectly suited for tabular datasets. It is like a list with each element having the same length, i.e. the column. It is the heterogeneous counterpart of a matrix since columns can be of different data type. Also within the same column elements can be of different types.

Create a dataframe

```
df <- data.frame(id = letters[1:5], A = 1:5, B = 6:10)
df <- data.frame(id = letters[1:5], A = c(1,'dos',TRUE,4,'cinque'), B = 6:10)
df
```

```
##   id      A  B
## 1  a      1  6
## 2  b    dos  7
## 3  c   TRUE  8
## 4  d      4  9
## 5  e cinque 10
```

```
#check if it is a list
is.list(df)
```

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

```
class(df)
```

```
## [1] "data.frame"
```

```
typeof(df)
```

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

Useful functions with dataframes

Some useful functions with dataframes follow.

```
#shows first n rows
head(df,n=2)
```

```
##   id   A B
## 1  a   1 6
## 2  b dos 7
```

```
#shows last n rows
tail(df,n=3)
```

```
##   id      A  B
## 3  c   TRUE  8
## 4  d      4  9
## 5  e cinque 10
```

```
#returns the dimensions of data frame (i.e. number of rows and number of columns)
dim(df)
```

```
## [1] 5 3
```

```
#number of rows
nrow(df)
```

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

```
#number of columns
ncol(df)
```

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

```
# structure of data frame: name, type and preview for each column
str(df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ id: Factor w/ 5 levels "a","b","c","d",..: 1 2 3 4 5
##  $ A : Factor w/ 5 levels "1","4","cinque",..: 1 4 5 2 3
##  $ B : int  6 7 8 9 10
```

```
#names or colnames() to show the names attribute for a data frame
names(df)
```

```
## [1] "id" "A"  "B"
```

```
colnames(df)
```

```
## [1] "id" "A"  "B"
```

```
#shows the class of each column in the data frame
sapply(df, class)
```

```
##        id         A         B
##  "factor"  "factor" "integer"
```

```
#the sapply() function takes list, vector or data frame as input and returns vector or matrix as output
```

Selecting elements

Similarly to the matrix case, both a row and a column indentifiers must be specified.

```
df[1, 3]
```

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

Since data frames are also lists, we can use the list notation: double square brackets or $.

The columns are in fact elements of such list.

```
df[["A"]]
```

```
## [1] 1      dos    TRUE   4      cinque
## Levels: 1 4 cinque dos TRUE
```

```
df$A
```

```
## [1] 1      dos    TRUE   4      cinque
## Levels: 1 4 cinque dos TRUE
```

Factors

A factor is a data structure specifically built for categorical variables that take on only a predefined number of values eg color = "yellow", "blue", "red". It can also be ordered eg humidity = 'low', 'medium', 'high'. These 2 examples have both 3 levels (unique categorical values).

Create a factor

```
#explicit level declaration
x <- factor(c("single", "married", "married", "married", "single"),
            levels = c("single", "married", "divorced"));
x
```

```
## [1] single  married married married single
## Levels: single married divorced
```

```
#if levels are not provided R infers them from the data provided
x2 <- factor(c("blue", "yellow", "red", "yellow", "blue"))
x2
```

```
## [1] blue   yellow red    yellow blue
## Levels: blue red yellow
```

```
#to add a level
levels(x) <- c(levels(x), "complicated")
x
```

```
## [1] single  married married married single
## Levels: single married divorced complicated
```

```
x[2]
```

```
## [1] married
## Levels: single married divorced complicated
```