Lecture 2: Atomic Data Types/Vectors

Wim R.M. Cardoen, PhD

Contents

1 R Objects **2**
1.1 Examples . . . 2

2 Atomic Data Types **4**
2.1 Examples . . . 4

3 Operations on atomic data types: **6**

4 Atomic vectors **7**
4.1 Creation of atomic vectors . . . 7
4.1.1 Examples . . . 7
4.2 Operations on vectors: element-wise . . . 9
4.2.1 Examples . . . 9
4.3 Retrieving elements of vectors . . . 9
4.3.1 Examples . . . 10
Other topics w.r.t vectors . . . 12
4.4 Alia . . . 12
Matrices & Arrays . . . 12
Exercises 3 . . . 12
Other topics on Data structures . . . 12
Conditionals & Loops . . . 13
Environments . . . 13
Functions . . . 13
Capita selecta . . . 13

R can be summarized in three principles (John M. Chambers, 2016)

- `Everything that exists in R is an object.`
- `Everything that happens in R is a function call.`
- `Interfaces to other languages are a part of R.`

1 R Objects

- An object in R is (internally) represented as a pair: (**symbol**, **value**).
- A **symbol** is assigned a **value** by the use of an arrow pointing to the left (<-).
- There are **less favored** ways:
  - A simple equality sign (=).
  - Using the **assign()** function.

1.1 Examples

- Clean up the global environment i.e. remove all objects from the current R environment. **STRONGLY RECOMMENDED!**

```
rm(list=ls())
ls()
```

```
## character(0)
```

- **preferred** way to assign variables

```
x <- 5.0
x
```

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

- alternative 1: mainly used to assign default function arguments

```
y = 5.0
y
```

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

```
mysamplevariance <- function(x, av=0){

  n <- length(x)
  if(n>1){
      return(1.0/(n-1)*sum((x-av)^2))
  }
  else{
      stop("ERROR:: Dividing by zero (n==1) || (n==0) ")
  }
}

x <- rnorm(10)
mysamplevariance(x)
```

```
## [1] 1.006127
```

```
mysamplevariance(x,mean(x))
```

```
## [1] 0.9276187
```

```
var(x)
```

```
## [1] 0.9276187
```

- alternative 2: even less used

```
assign("z", 5.0)
z
```

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

- functions are objects

```
f <- mean
f
```

```
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x55f9e87def90>
## <environment: namespace:base>
```

```
val <- f(1:10)
val
```

```
## [1] 5.5
```

2 Atomic Data Types

`Nothing exists except atoms and empty space; everything else is opinion.  (Democritos)`

- R has the following 6 **atomic** data types:
  - logical (i.e. boolean)
  - integer
  - double
  - character (i.e. string)
  - complex
  - raw (i.e. byte)

The latter 2 types (i.e. complex and especially raw) are less common.

The **typeof()** function determines the **INTERNAL** storage/type of an R object.

2.1 Examples

- boolean/logical values: either **TRUE** or **FALSE**

```
x1 <- TRUE
x1
```

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

```
typeof(x1)
```

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

- integer values ($\in \mathbb{Z}$):

```
x2 <- 3L
x2
```

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

```
typeof(x2)
```

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

- double (precision) values:

```
x3 <- 3.14
x3
```

```
## [1] 3.14
```

```
typeof(x3)
```

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

- character values/strings

```
x4 <- "Hello world"
x4
```

```
## [1] "Hello world"
```

```
typeof(x4)
```

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

- complex values ($\in \mathbb{C}$):

```
x5 <- 2.0 + 3i
x5
```

```
## [1] 2+3i
```

```
typeof(x5)
```

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

3 Operations on atomic data types:

- logical operators: ==, !=, &&, ||, !
- numerical operators: +, -, *, /, ^, **
- but also:
  - integer division: %/%
  - modulo operation: %%
  - **Note**: matrix multiplication will be performed using %*%
- string manipulation:
  - nchar():
  - paste()
  - cat()
  - sprintf()
  - substr
  - strsplit()
  - **Note**: Specialized R libraries were developed to manipulate strings e.g. *stringr*
- explicit casts/conversion: https://data-flair.training/blogs/r-string-manipulation/
  - as.{logical, integer, double, character, complex}()
- explicit tests of the type of a variable:
  - is.{logical, integer, double, character, complex}()

4 Atomic vectors

- An **atomic** vector is a data structure containing elements of **only one atomic** data type. Therefore, an atomic vector is **homogeneous**.
- Atomic vectors are stored in a **linear** fashion.
- R does **NOT** have scalars:
  - An atomic vector of **length 1** plays the role of a scalar.
  - Vectors of **length 0** also exist (and they have some use!).
- A **list** is a vector not necessarily of the atomic type.
  A list is also known as a **recursive/generic** vector (*vide infra*).

4.1 Creation of atomic vectors

Atomic vectors can be created in a multiple ways:

- Use of the **vector()** function.
- Use of the **c()** function (**c** stands for concatenate).
- Use of the column operator **:**
- Use of the **seq()** and **rep()** functions.

The length of a vector can be retrieved using the **length()** function.

4.1.1 Examples

```
# Create empty vector (Default mode:"logical")
x <- vector(); x
```

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

```
length(x)
```

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

```
typeof(x)
```

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

```
x <- vector(mode="complex", length=4) ;x
```

```
## [1] 0+0i 0+0i 0+0i 0+0i
```

```
length(x)
```

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

```
x
```

```
## [1] 0+0i 0+0i 0+0i 0+0i
```

```
x[1] <- 4
x
```

```
## [1] 4+0i 0+0i 0+0i 0+0i
```

```
# Concatenate operator
x1 <- c(3, 2, 5.2, 7) ; x1
```

```
## [1] 3.0 2.0 5.2 7.0
```

```r
x2 <- c(8, 12, 13) ; x2
```

```
## [1]  8 12 13
```

```r
x3 <- c(x2, x1) ; x3
```

```
## [1]  8.0 12.0 13.0  3.0  2.0  5.2  7.0
```

```r
x4 <- c(FALSE,TRUE,FALSE) ; x4
```

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

```r
x5 <- c("Hello", "Salt", "Lake", "City") ; x5
```

```
## [1] "Hello" "Salt"  "Lake"  "City"
```

```r
# Use of the column operator
y1 <- 1:10 ; y1
```

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

```r
y2 <- 5:-5 ; y2
```

```
##  [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
```

```r
y3 <- 2.3:10 ; y3
```

```
## [1] 2.3 3.3 4.3 5.3 6.3 7.3 8.3 9.3
```

```r
y4 <- 2.0*7:1 ; y4
```

```
## [1] 14 12 10  8  6  4  2
```

```r
y5 <- 1:7-1 ; y5
```

```
## [1] 0 1 2 3 4 5 6
```

```r
# The seq function
z1 <- seq(from=1, to=15, by=3); z1
```

```
## [1]  1  4  7 10 13
```

```r
z2 <- seq(from=-2,to=5,length=4); z2
```

```
## [1] -2.0000000  0.3333333  2.6666667  5.0000000
```

```r
# The rep function
z3 <- rep(c(3,2,4), time=2); z3
```

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

```r
z4 <- rep(c(3,2,4), each=3); z4
```

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

```r
z5 <- rep(c(1,7), each=2, time=3) ; z5
```

```
##  [1] 1 1 7 7 1 1 7 7 1 1 7 7
```

```r
length(z5)
```

```
## [1] 12
```

4.2 Operations on vectors: element-wise

- All operations on vectors in R happen **element by element** (cfr. *NumPy*).
- **Vector Recycling**:

  If 2 vectors of **different** lengths are involved in an operation, the **shortest vector** will be repeated until all elements of the longest vector are matched.
  A message will be sent to the stdout.

4.2.1 Examples

```
# Operations on vectors :: element-wise
x <- -3:3; x
```

```
## [1] -3 -2 -1  0  1  2  3
```

```
y <- 1:7; y
```

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

```
xy <- x*y; xy
```

```
## [1] -3 -4 -3  0  5 12 21
```

```
xpy <- x^y; xpy
```

```
## [1]   -3    4   -1    0    1   64 2187
```

```
# R: allows recycling
x <- 0:10
y <- 1:2
length(x)
```

```
## [1] 11
```

```
length(y)
```

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

```
x
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
y
```

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

```
x+y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

```
##  [1]  1  3  3  5  5  7  7  9  9 11 11
```

4.3 Retrieving elements of vectors

- Indexing: starts at **1** (**not 0** like C/C++, Python, Java, ...) see also:
  Edsger Dijkstra: Why numbering should start at zero
- Use of vector with indices to extract values.
- Advanced features:

- use of boolean values to extract values.
- the membership operator: **%in%**.
- the deselect/omit operator: **-**
- **which()**: returns the indices for which the condition is true.
- **any()**/**all()** functions.
  - **any()** : **TRUE** if at least 1 value is true
  - **all()** : **TRUE** if all values are true

4.3.1 Examples

- Use of a simple index:

```
x <- seq(2,100,by=15)
x[4]
```

```
## [1] 47
```

```
x[1]
```

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

- Select several indices at once using vectors:

```
x
```

```
## [1]  2 17 32 47 62 77 92
```

```
x[3:5]
```

```
## [1] 32 47 62
```

```
x[c(1,3,5,7)]
```

```
## [1]  2 32 62 92
```

```
x[seq(1,7,by=2)]
```

```
## [1]  2 32 62 92
```

- Extraction via booleans (i.e. retain only those values that are equal to **TRUE**):

```
x
```

```
## [1]  2 17 32 47 62 77 92
```

```
x>45
```

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

```
x[x>45]
```

```
## [1] 47 62 77 92
```

- Use of the **%in%** operator:

```
x
```

```
## [1]  2 17 32 47 62 77 92
```

```
10 %in% x
```

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

```
62 %in% x
```

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

```
c(32,33,43) %in% x
```

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

```
!(c(32,33,43) %in% x)
```

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

- Negate/filter out the elements with **negative** indices:

```
x
```

```
## [1]  2 17 32 47 62 77 92
```

```
x[-c(2,4,6)]
```

```
## [1]  2 32 62 92
```

```
z <- x[-1] - x[-length(x)]
z
```

```
## [1] 15 15 15 15 15 15
```

- The **which()** function returns **only those indices** of which the condition/expression is **true**.

```
# Sample 10 numbers from N(0,1)
vecnum <- rnorm(n=10)
vecnum
```

```
##  [1] -0.3158073 -1.6773463  0.4756900 -1.0435805  1.5402168 -0.1069257
##  [7] -0.6741653 -0.3925438 -0.5771441  0.1899638
```

```
which(vecnum>1.0)
```

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

- Use of the **any()**/**all()** functions.

```
y <- seq(0,100,by=10)
x
```

```
## [1]  2 17 32 47 62 77 92
```

```
y
```

```
##  [1]   0  10  20  30  40  50  60  70  80  90 100
```

```
any(x<y)
```

```
## Warning in x < y: longer object length is not a multiple of shorter object
## length
```

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

```
all(x[6:7]>y[2:3])
```

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

Other topics w.r.t vectors

- Coercion (implicit & explicit) # STILL TO be done
- Hash Table
- Presence of NA

4.4 Alia

- boolean vector vs unique value

Matrices & Arrays

- Attributes
- Matrices & arrays
- Matrix multiplication

Exercises 3

- Special types:
  - Factors
  - Date
  - Time
- NA, NaN, NULL

- Logical operators:
  - && vs. &.
  - || vs. |.
  - xor()

Other topics on Data structures

- List
- Dataframe & Tibble
- IO (read.csv, read.file)
- Names

- Subsetting, [[]] vs. []

Conditionals & Loops

- if, else, else if switch and elseif
- for
- while
- repeat
- return()

Environments

- search(), attach, detach
- library

Functions

- lexical scoping
- simple functions
- args(), formals()
- default arg, ...
- lazy evaluation
- closure
- anonymous functions
- make your own operators
- loop functions: {l,s,m}apply, split

Capita selecta

- profiling, debugging