Professor Bear - R Data structures

**Table of Contents**  
[Section 1.1: Data structures in R](#data-structures-in-r)  
[Section 1.1.1: Vectors](#vectors)  
[Section 1.1.1: Lists](#lists)  
[Section 1.1.1: Arrays](#arrays)  
[Section 1.1.1: Matrices](#matrices)  
[Section 1.1.1: Data frames](#data-frames)

## 1.1: Data structures in R

Data Structures are the programmatic way of storing data so that data can be used efficiently. One can think of a data structure as a container for data.

## 1.1.1: Vectors

Usually when you create variables in R, you create **vectors**. A vector is simply a set of elements *of the same class* (e.g. character, numeric, integer, or logical -as in True/False). It is the basic data structure in R. Most commonly, you will use the c() function (c stands for concatenate) to create vectors. :

v1 <- c(1,2,3,4,5) #creates a numeric vector  
v1

## [1] 1 2 3 4 5

v2 <- c(1L, 2L) #creates an integer vector  
v2

## [1] 1 2

v3 <- c(TRUE, FALSE) #creates a logical vector  
v3

## [1] TRUE FALSE

v4 <- c("a", "b", "c") #creates a character vector  
v4

## [1] "a" "b" "c"

v5 <- c(1+0i, 3+5i) #creates a complex vector   
v5

## [1] 1+0i 3+5i

*Operations on vectors*

Once you have a vector (or a list of numbers) in memory most basic operations are available. This makes R very powerful.

v1

## [1] 1 2 3 4 5

v1 + 5 # add 5 to each of the numbers

## [1] 6 7 8 9 10

v1\*3 # Multiply each of the numbers by 3

## [1] 3 6 9 12 15

v1/3 # Divide each of the numbers by 3

## [1] 0.3333333 0.6666667 1.0000000 1.3333333 1.6666667

log(v1) # Take the log of each of the numbers

## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379

We can even add vectors to vectors.

a <- c(1,2,3)  
a

## [1] 1 2 3

b <- c(1,2,3,4,5)  
b

## [1] 1 2 3 4 5

a+b

## Warning in a + b: longer object length is not a multiple of shorter object  
## length

## [1] 2 4 6 5 7

If the lengths of the vectors differ then you may get an error message, a warning message and unpredictable results. It is best if they are the same length.

a <- c(1,2,3)  
a

## [1] 1 2 3

length(a)

## [1] 3

b <- c(3,4,5)  
b

## [1] 3 4 5

length(b)

## [1] 3

a+b

## [1] 4 6 8

If you mix in a vector elements that are of a different class (for example numerical and logical), R will **coerce** to the minimum common denominator, so that every element in the vector is of the same class.

b <- c(33, "bear")  
class(b)

## [1] "character"

Note what happens when a vector is placed in another vector.

x <- c(2, 3, 5)  
x

## [1] 2 3 5

class(x)

## [1] "numeric"

y <- c(6, x, 2)  
y

## [1] 6 2 3 5 2

class(y)

## [1] "numeric"

## 1.1.1: Lists

A list is a generic data structure containing other objects. Unlike a vector it allows us to create a mixed data sequence.

w <- c(1, 2, 3)   
w

## [1] 1 2 3

class(w)

## [1] "numeric"

x <- c("a", "b", "c", "d", "e")   
x

## [1] "a" "b" "c" "d" "e"

class(x)

## [1] "character"

y <- c(TRUE, FALSE, TRUE, FALSE)   
y

## [1] TRUE FALSE TRUE FALSE

z <- list(w, y, x, 33, "bear", FALSE) # create a mixed data type list  
z

## [[1]]  
## [1] 1 2 3  
##   
## [[2]]  
## [1] TRUE FALSE TRUE FALSE  
##   
## [[3]]  
## [1] "a" "b" "c" "d" "e"  
##   
## [[4]]  
## [1] 33  
##   
## [[5]]  
## [1] "bear"  
##   
## [[6]]  
## [1] FALSE

class(z)

## [1] "list"

*List Slicing*

We use the single square bracket "[]" operator to access elements at the first level of a list. The index 1 is the first element. We use the doubkle square bracket "[]" operator to access elements at the second level, etc.

z

## [[1]]  
## [1] 1 2 3  
##   
## [[2]]  
## [1] TRUE FALSE TRUE FALSE  
##   
## [[3]]  
## [1] "a" "b" "c" "d" "e"  
##   
## [[4]]  
## [1] 33  
##   
## [[5]]  
## [1] "bear"  
##   
## [[6]]  
## [1] FALSE

z[1] # First element

## [[1]]  
## [1] 1 2 3

z[[2]] # cond element

## [1] TRUE FALSE TRUE FALSE

z[[2]][1] # First element of the second element

## [1] TRUE

## 1.1.1: Arrays

Vectors are one-dimensional arrays in R and matrices are two-dimensional arrays in R. We can create n-dimensional arrays as a set of stacked matrices of identical dimensions. This will be discuss in the [matrices](#matrices) section below. The term arrays is discussed here becuase most programming languages use the term array. The reason R uses the term vector, is the basic operations can be applied to vectors whereas most programming languages use functions and loops to apply operations to arrays.

## 1.1.1: Matrices

Matrices are n-dimensional vectors (usually two-dimensional). We build matrices from vectors (one-dimensional arrays). You can create a matrix in two ways. By using the command *matrix*.

x<-matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)  
x

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

class(x)

## [1] "matrix"

Or using *cbind()* or *rbind()* to add columns or rows to a vectors.

x <- c(1, 2, 3) # Creates a vector `x' of 3 values.  
x

## [1] 1 2 3

class(x)

## [1] "numeric"

y <- c(55, 33, 11) # Creates another vector `y' of 3 values.  
y

## [1] 55 33 11

class(y)

## [1] "numeric"

a<-rbind(x, y) # Creates a 2 x 3 matrix. Note that the rows are appended  
a

## [,1] [,2] [,3]  
## x 1 2 3  
## y 55 33 11

class(a)

## [1] "matrix"

b<-cbind(x, y) # Creates a 3 x 2 matrix. Note that the columns are appended  
b

## x y  
## [1,] 1 55  
## [2,] 2 33  
## [3,] 3 11

class(b)

## [1] "matrix"

*Matrix element access*

As with vectors, square brackets extract specific values from a matrix but more values are used within the brackets seperated by commas for each diminasion.

b

## x y  
## [1,] 1 55  
## [2,] 2 33  
## [3,] 3 11

b[,2] # Extracts the second column

## [1] 55 33 11

b[1,] # Extracts the first row

## x y   
## 1 55

b[1,2] # Extracts element in the first row and the second column

## y   
## 55

b[1,2]<=55 # Recplaces the element in the first row and the second column with 55

## y   
## TRUE

b

## x y  
## [1,] 1 55  
## [2,] 2 33  
## [3,] 3 11

b[1,] <- c(3,3) # Replaces the first row   
b

## x y  
## [1,] 3 3  
## [2,] 2 33  
## [3,] 3 11

*Matrix operations and functions*

R supports a variety of matrix functions, including: det(), which returns the matrix's determinant; t(), which transposes the matrix; solve(), which inverts the the matrix; dim() command returns the dimensions of your matrix.

b

## x y  
## [1,] 3 3  
## [2,] 2 33  
## [3,] 3 11

dim(b)

## [1] 3 2

## 1.1.1: Data frames

A data frame is used for storing data tables of mixed data type. Typically, data from an excel sheet will be imported in to R as a data frame. We will use the built-in data set “InsectSprays” to discuss data frames.

data(InsectSprays)  
names(InsectSprays)

## [1] "count" "spray"

head(InsectSprays,3)

## count spray  
## 1 10 A  
## 2 7 A  
## 3 20 A

dim(InsectSprays)

## [1] 72 2

nrow(InsectSprays)

## [1] 72

ncol(InsectSprays)

## [1] 2

levels(InsectSprays$spray)

## [1] "A" "B" "C" "D" "E" "F"

summary(InsectSprays)

## count spray   
## Min. : 0.00 A:12   
## 1st Qu.: 3.00 B:12   
## Median : 7.00 C:12   
## Mean : 9.50 D:12   
## 3rd Qu.:14.25 E:12   
## Max. :26.00 F:12

## 1.17: Further resources

[LearnR](https://youtu.be/p3i7Kz6C_-4?list=PLFAYD0dt5xCwDNFdrqeNoU9t-nhAWkbKe)

[Try R @codeschool](<http://tryr.codeschool.com>)

[Datacamp R Tutorials](https://www.datacamp.com/)

[rstudio online learning](https://www.rstudio.com/online-learning/)