Chapter 1

R objects - Part 1

1.1 Objects

R is said to be object oriented. This means that everyting in R is an object. That is not so strange when we think about a single number or a vector. It may however become more diffuse when an object can include other objects or when all results from a statistical calculation or even a plot is an object. We start from the beginning with numbers and vectors of numbers.

1.2 Numbers

R can e.g. be used as a calculator (some examples below). We can also give names to the objects. Upper case matters.

```
4+13

[1] 17

21-4

[1] 17

21-(-4)

[1] 25

5*7

[1] 35

-3/2

[1] -1.5

5^2 # 5 squared

[1] 25

2^4 # ^ is the power function

[1] 16

9^0.5 # the square root
```

```
[1] 3

sqrt(9) # the specific function for square root

[1] 3

        # (more about functions later)

2^(-2) # 1/(2*2)

[1] 0.25

x<-5
x

[1] 5

Important.Number<-14
Important.Number

[1] 14

x*Important.Number

[1] 70

Important.Number/x

[1] 2.8

x<-2    # it is easy to change x
1/x+1/x^2+1/x^3+1/x^4+1/x^5

[1] 0.96875
```

1.3 Vectors

Vectors play an important role in data analysis because the data in each of our variables *is* a vector. A vector is a list of data which are all of the same type e.g. numbers, character strings or factors. Data frames are the most common data representation when we do statistical analyses or other calculations. We can use vectors to construct dataframes or we can extract vectors from dataframes.

1.3.1 Contruction of vectors - the c( ) function

The function `c(some comma-separated list of object arguments) combines` the arguments into a single new object (if possible!). It is one of **the most widely used of all R functions** and is used very often when you want to use a group of numbers or characters as an argument in some other function. Its murky origins in Unix, mean that it is also sometimes referred to as *concatenate*.

If you use `c()` to combine a list of arguments which have different classes, R tries to figure out how the user might want to combine the arguments with a common type. (Remember our standard *atomic vector* must have all elements of the same class.)

Here are some examples:

1.3.2 Examples of numerical vectors and calculations

Now we use vectors of numbers.

```
x<-c(2,4,6)
x

[1] 2 4 6

2*x

[1]  4  8 12

y<-x*x
y

[1]  4 16 36

x^2

[1]  4 16 36

z<-(x+y)/2
z

[1]  3 10 21

z<-c(z,x,3,14)
z

[1]  3 10 21  2  4  6  3 14

xi<-5:12  # to generate a sequence
xi

[1]  5  6  7  8  9 10 11 12
```

1.4 Type and class

R objects have both **type** and **class**. The objects of numbers and vectors above have type=double and class=numeric. Most often this is not necessary to have in mind. However the function **str** is useful to assess the structure of an object as well as the functions **typeof** and **class**.

```
x<-c(2,4,6)
typeof(x) # real numbers

[1] "double"

class(x)

[1] "numeric"

str(x)    # decribe the object

 num [1:3] 2 4 6
```

1.4.1 Integer

Can be described as numerical without decimals

```
x<-c(2,4,6)
y<-as.integer(x)
x

[1] 2 4 6

y

[1] 2 4 6

typeof(x)

[1] "double"

class(x)

[1] "numeric"

typeof(y) #even if you cannot see the difference x and y have different type

[1] "integer"

class(y)

[1] "integer"
```

1.4.2 Logical

The elements of the vector are either TRUE or FALSE.

```
z1<-c(TRUE,FALSE,TRUE)
z1

[1]  TRUE FALSE  TRUE

typeof(z1) # this is an important type in R

[1] "logical"

class(z1)

[1] "logical"

z2<-c(T,F,T) # the same as z1
z2

[1]  TRUE FALSE  TRUE

y<-z1*1 # converting logical to a so called dummy variable
y

[1] 1 0 1

5<7                           # The result is logical

[1] TRUE
```

```
x<-c(5<7,5>7,2==2)            # "==" means equal to
x

[1]  TRUE FALSE  TRUE

# some more examples (!= means not equal to)
x<-c(5>5, 5>=5,3!=5,4!=4)
x

[1] FALSE  TRUE  TRUE FALSE

!x

[1]  TRUE FALSE FALSE  TRUE
```

1.4.3 Character

The elements are text strings.

```
x<-"b"
x

[1] "b"

y<-"R-object"
y

[1] "R-object"

x<-c(x,y,"c","another phrase")
x

[1] "b"              "R-object"       "c"              "another phrase"

typeof(x)

[1] "character"

class(x)

[1] "character"
```

Own experimentation

Try creating numeric vectors and do calculations. Also create vectors of other type like character and logical.

1.5 Accessing elements of a vector

Elements of a vector are accessed by giving the row(s) within brackets [ ] or using a logical vector of the same length.

```
x<-c(2,4,7,13,5,8)
x
```

```
[1]  2  4  7 13  5  8

y<-x[3]
y

[1] 7

z<-x[c(1,4,6)]
z

[1]  2 13  8

y2<-x[c(1,1,4,4,4,6,6,6,6,6)]
y2

 [1]  2  2 13 13 13  8  8  8  8  8

z2<-x[-c(2,3,5)] # minus sign means exclusion of the elements
z2

[1]  2 13  8

x2<-x[c(T,F,F,T,F,T)] # you can also use a vector (of the same length) of logical elements
x2

[1]  2 13  8
```

1.6 Concatenating text elements with the paste function

It is sometimes useful to concatenate text and numerical elements to a string.

```
paste("part one","part two") # one string

[1] "part one part two"

# Functions usually have arguments. For paste one argument is "sep"
paste("part one","part two",sep="")  # sep=" " is the default

[1] "part onepart two"

paste0("part one","part two")  # alternative wrapper function with sep="" as the default

[1] "part onepart two"

paste("part one","part two",sep=" | ")

[1] "part one | part two"

x<-c(0.49,"Pearson's r")
x

[1] "0.49"        "Pearson's r"

paste("The result from the calcultaion of",x[2],"=",x[1])

[1] "The result from the calcultaion of Pearson's r = 0.49"
```

Own experimentation

Try to create vectors and access its elements. You may also try the paste function.

1.7 Matrixes

A matrix is a two-dimensional vector with n rows and k columns. More abut matrix calculations later.

```
x<-seq(1,16)
y<-matrix(x,nrow=4)
y

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

typeof(y)

[1] "integer"

class(y)

[1] "matrix" "array"

z<-matrix(x,ncol=8)
z

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    3    5    7    9   11   13   15
[2,]    2    4    6    8   10   12   14   16

# data are stored by column
```

1.8 Data frames

A data frame is a list of variables. However, **list** is a more general object which will be presented later. Each variable is a vector of equal length. The elements of the list are the columns of the data frame. The types of these vectors (the data frame columns) are not required to be the same, though, of course, within the columns each element must be of the same type.

Data frames can access columns using df$colName notation and they can also use matrix-style notation. These are both demonstrated below.

Where possible, it is recommended that you use the df$colName notation because it is easier to follow in scripts and also if column order is changed, the named notation still works. As you might expect, a data frame can be constructed from a group of equal length vectors.

```
people <- c("Lena", "Solveig", "Anna", "Hans", "Erik" )
byear <-  c(1982,      1976,      1949,      2001,    1967)
score <-  c( 22,         43,        87,        45,      60)
```

```
salary <- c(40,          47,         31,         32,       45)
df <- data.frame(Name = people, BirthYear = byear, Score = score, Salary = salary)
# New variable names are given for the data frame
df

     Name BirthYear Score Salary
1    Lena      1982    22     40
2 Solveig      1976    43     47
3    Anna      1949    87     31
4    Hans      2001    45     32
5    Erik      1967    60     45

df$Sex<-c("woman","woman","woman","man","man")  # we can add a variable
df

     Name BirthYear Score Salary   Sex
1    Lena      1982    22     40 woman
2 Solveig      1976    43     47 woman
3    Anna      1949    87     31 woman
4    Hans      2001    45     32   man
5    Erik      1967    60     45   man

typeof(df)

[1] "list"

class(df)

[1] "data.frame"

# A matrix can be converted to a data frame
y

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

ydf<-data.frame(y)
ydf        # no names

  X1 X2 X3 X4
1  1  5  9 13
2  2  6 10 14
3  3  7 11 15
4  4  8 12 16

names(ydf)<-c("Var1","Var2","Var3","Var4")
ydf

  Var1 Var2 Var3 Var4
1    1    5    9   13
2    2    6   10   14
3    3    7   11   15
4    4    8   12   16
```

NOTE that this may be a little confusing to look at, because the variables (vectors) have been laid out across the page (rows) and in the dataframe they are placed in columns.

1.8.1 Accessing elements of a data frame

Here are the common ways of accessing elements of the dataframe

```
df$BirthYear[3] # List form - 3rd row of column "BirthYear"

[1] 1949

df$byear        # does not work

NULL

df[2,3]         # Matrix form - 2nd row of 3rd column

[1] 43

df[,3]          # Row number omitted, assume ALL rows i.e. this is equivalent to df$score

[1] 22 43 87 45 60

df[4:5,1:3]     # Selects the subset - rows 4 and 5, columns 1  to 3

  Name BirthYear Score
4 Hans      2001    45
5 Erik      1967    60

df[5:1,]            # reverse order

     Name BirthYear Score Salary   Sex
5    Erik      1967    60     45   man
4    Hans      2001    45     32   man
3    Anna      1949    87     31 woman
2 Solveig      1976    43     47 woman
1    Lena      1982    22     40 woman

df[c(2,5,3,1,4),]   # specific order

     Name BirthYear Score Salary   Sex
2 Solveig      1976    43     47 woman
5    Erik      1967    60     45   man
3    Anna      1949    87     31 woman
1    Lena      1982    22     40 woman
4    Hans      2001    45     32   man

df[c(5,1,2),]       # sorting and selection combined

     Name BirthYear Score Salary   Sex
5    Erik      1967    60     45   man
1    Lena      1982    22     40 woman
2 Solveig      1976    43     47 woman

df[c(1,2,3,2,3),]   # repeating rows

       Name BirthYear Score Salary   Sex
1      Lena      1982    22     40 woman
2   Solveig      1976    43     47 woman
3      Anna      1949    87     31 woman
2.1 Solveig      1976    43     47 woman
3.1    Anna      1949    87     31 woman

df[,c(1,2,3,2,3)]   # also possible with columns
```

```
     Name BirthYear Score BirthYear.1 Score.1
1    Lena      1982    22        1982      22
2 Solveig      1976    43        1976      43
3    Anna      1949    87        1949      87
4    Hans      2001    45        2001      45
5    Erik      1967    60        1967      60

df[c(T,F,T,T,F),]    # selection due to logical vector of same length (nrow)

  Name BirthYear Score Salary   Sex
1 Lena      1982    22     40 woman
3 Anna      1949    87     31 woman
4 Hans      2001    45     32   man
```

The last example is a basis for filtering. More about that later.

Own experimentation

Create a data frame (don't make it complicated, use simple elements like numbers and short text strings) and try to access specific rows or columns from it.

1.9 Tibbles

Tibbles are the tidyverse equivalent (almost) of a dataframe, but they avoid some of the problems - like if you name a large dataframe in the Console, you will fill the pane with numbers. A tibble will only print the first ten row and as many columns as will fit on the screen. The printout shows the variable type, too. If you really want to see all your variables, you can use `print` and set the `width` to `Inf`.

The datasets available in the core release of R are generally data.frames. Dataframes can easily be converted to tibbles and vice versa. Finally a warning - occasionally older R functions won't work with tibbles so you may need to convert your tibble back to a data.frame.

```
library(tibble)
df <- as_tibble(df)
class(df)

[1] "tbl_df"     "tbl"        "data.frame"

df <- as.data.frame(df)
class(df)

[1] "data.frame"
```