Introduction to R programming for data science – day 2

Dr. Francesca Finotello
Medical University of Innsbruck, Austria

Accessing and subsetting objects

Accessing vectors

You can access the elements of a vector by using *logical* or *numeric* indexes (positive and negative) between brackets:

```
days <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")

days[c(1,6,7)]
```

```
## [1] "Mon" "Sat" "Sun"
```

```
weekend <- days[c(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE)]

weekend <- days[-seq(1,5)] # Negative ind. of elements to be removed
```

Accessing vectors

To access vectors, you must use square brackets.
Round brakets are for functions.

```
age <- c(11, 12, 18, 20, 45, 2, 33)

age[1]
```

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

```
age(1)
```

```
## Error in age(1): could not find function "age"
```

```
mean(age)
```

```
## [1] 20.14286
```

Accessing vectors

Vector elements can be also accessed using their names, when initialized

```
age <- c(11, 12, 18, 20)
names(age) <-c("John", "Lisa", "Maria", "Markus")

age
```

```
##   John   Lisa  Maria Markus
##     11     12     18     20
```

```
age[c("John", "Maria")]
```

```
##  John Maria
##    11    18
```

Subsetting vectors

You can access the elements of a vector that satisfy a certain condition:

```
age <- c(11, 12, 18, 20, 45, 2, 33)

age[age>=18] # Logical index
```

```
## [1] 18 20 45 33
```

```
age[which(age>=18)] # Numerical index
```

```
## [1] 18 20 45 33
```

What is the difference between age>=18 and which(age>=18)?

Manipulating vectors

You can change the elements of a vector:

```
x <- c(11, 12, -18, 20, -45, -2, 33)

x[1] <- 35 # Subsitute the first element

x
```

```
## [1]  35  12 -18  20 -45  -2  33
```

```
x[x<0] <- 0 # Set to 0 all negative elements

x
```

```
## [1] 35 12  0 20  0  0 33
```

Accessing matrices

```
M <- matrix(1:6, nrow=2, byrow=FALSE)
colnames(M) <- c("Sample_A", "Sample_B", "Sample_C")
rownames(M) <- c("Gene1", "Gene2")
M
```

```
##       Sample_A Sample_B Sample_C
## Gene1        1        3        5
## Gene2        2        4        6
```

```
M[1,2] # Two indices: [row,column]
```

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

Accessing matrices

Accessing the first row:

```
M[1,]
```

```
## Sample_A Sample_B Sample_C
##        1        3        5
```

Removing the second column:

```
M[,-2]
```

```
##       Sample_A Sample_C
## Gene1        1        5
## Gene2        2        6
```

Accessing matrices

Elements of matrices and data.frames can be accessed also considering row and column names

```
M["Gene1",]
```

```
## Sample_A Sample_B Sample_C
##        1        3        5
```

```
M[,"Sample_C"]
```

```
## Gene1 Gene2
##     5     6
```

Subsetting matrices: the "drop" option

By default, R transforms one-dimentional objects into vectors, unless we specify drop=FALSE

```
v <- M[1,]

dim(v)
```

```
## NULL
```

```
m <- M[1,, drop=FALSE]

dim(m)
```

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

Accessing/subsetting data.frames

Data.frames can be handled similarly to matrices

```
( DF <- data.frame(name=c("Mary", "John"), age=c(19, 30),
  stringsAsFactors=FALSE) )
```

```
##   name age
## 1 Mary  19
## 2 John  30
```

```
DF[,"age"]
```

```
## [1] 19 30
```

```
DF[1,2]
```

```
## [1] 19
```

Accessing/subsetting lists

Lists can be accessed using double square brackets

```
favorites <- list(colors=c("blue", "purple"),
  cities=c("Venice", "Innsbruck", "New York City"))

favorites[[1]]
```

```
## [1] "blue"   "purple"
```

```
favorites[["cities"]]
```

```
## [1] "Venice"        "Innsbruck"     "New York City"
```

Exercises

Ex. 2.1

```
x <- c(0, -1, 3, 10, -14, 7.5, 9)
```

Save in vector *y* only the non-negative elements of *x* by using:

- Negative indexes
- Positive indexes
- A logical vector
- A “rule” using > or ≤

Ex. 2.2

```
normExpr <- c(10.2, 11.4, 4.0)
names(normExpr) <- c("CD8A", "CD8B", "PDCD1")
```

Access the expression of the CD8A and CD8B genes by using:

- Positive indexes
- Vector names

Ex. 2.3

```
y <- c(11, 12, -4, 7, 0)
z <- y>0
```

What is the length of *z*?

1. 3
2. 4
3. 5
4. None of the above

Ex. 2.4

```
cellFractions <- c(-0.1, 0.4, -0.4, 0.5, 0.2)
```

Set to 0 all negative cell fractions.

Ex. 2.5

```
M <- matrix(1:6, nrow=3, byrow=FALSE)
colnames(M) <- c("Sample_A", "Sample_B")
rownames(M) <- c("Gene1", "Gene2", "Gene3")
```

Save the second row of matrix *M* into:

- A vector called *v*
- A 1x2 matrix called *N*

Try to use both:

- A numeric index
- The matrix row names

Ex. 2.6

```
DF <- data.frame(name=c("Mary", "John"),
    age=c(19, 30))

x <- DF[,"name"]
```

What is the class of *x*?

1. Character
2. Numeric
3. Factor
4. Logical

Functions

Functions

A function is a set of statements organized together to perform a specific task.

R has a large number of built-in functions.

Additionally, users can:

- Import *R packages* enclosing a set of functions for specific tasks (e.g. differential gene expression analysis from RNA sequencing data)
- Create *new functions*.

Functions

Functions are usually invocated with their name, followed by round brakets listing the arguments to be considered (if any).

```
v1 <- c(1, 3, 56, 6)
v2 <- c(4, 5, 40, 7)
cor(x=v1, y=v2, method="pearson")
```

```
## [1] 0.999966
```

The name of the arguments can be omitted if the right order of the arguments is respected:

```
cor(v1, v2, method="pearson")
```

```
## [1] 0.999966
```

Functions

So far, we have used already several functions:

log10, sqrt, class, is.na, c, dim, ncol, nrow, colnames, rownames, factor, names, length, levels, as.factor, as.numeric, list, data.frame, which, cor.

There are other arithmetical functions:

sum, prod, max, min, which.max, which.min, range, median, var, sd, round, sign, exp.

What is the difference between max and which.max? Build a numeric vector and check!

The help function

If we do not know how a function works, we can invocate the help function to understand:

- Which task it performs
- Which are its arguments, their expected order, and their default values
- Which results it generates (e.g. a plot or an object with a specific format)
- If there are similar functions
- How it can be used, explained through examples

```
help(mean)

help(sample)
```

Useful functions: length and nchar

```
x <- c("Schildkroete",
  "Ohrwurm",
  "Rechtsschutzversicherungsgesellschaften")

length(x) # Length of the vector of strings
```

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

```
nchar(x) # Length of the strings
```

```
## [1] 12  7 39
```

Useful functions: table and unique

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

unique(x) # Unique elements of x
```

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

```
table(x) # Occurrences of each value in x
```

```
## x
## a b c d e
## 1 1 2 1 3
```

Useful functions: order and sort

```
x <- c(10, 2, 20, -1)

sort(x) # Sorted x
```

```
## [1] -1  2 10 20
```

```
order(x) # Indexes of sorted x elements
```

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

Useful functions: match

```
names <- c("Maria", "Markus", "Lena")
sel <- c("Lena", "Maria")

match(sel, names) # Find 'sel' elements in 'names'
```

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

Useful functions: rbind and cbind

```
(M1 <- matrix(1, ncol=2, nrow=2))
```

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

```
(M2 <- matrix(2, ncol=2, nrow=2))
```

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

Useful functions: rbind and cbind

```
cbind(M1,M2)
```

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

```
rbind(M1,M2)
```

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

Useful functions: apply

```
( M <- rbind(c(1,1,1), c(2,2,2)) )
```

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

```
apply(M,1,sum)
```

```
## [1] 3 6
```

```
apply(M,2,sum)
```

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

Useful functions: head and tail

```
M3 <- rbind(M1,M2)
head(M3,2)
```

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

```
tail(M3,2)
```

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

```
head(seq(1,100), 10)
```

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

33/52

Useful functions for sets

```
A <- c(1, 2, 3, 4, 5, 6, 6); B <- c(2, 4, 6, 8, 10)
union(A,B)
```

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

```
intersect(A,B)
```

```
## [1] 2 4 6
```

```
setdiff(A,B)
```

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

What would be the result of setdiff(B,A)?

Function: Hello World!

```
helloWorld <- function() {

  outmessage <- "Hello World!\n"

  cat(outmessage)

}
```

Function: Hello World!

```
helloWorld <- function() {

  outmessage <- "Hello World!\n"

  cat(outmessage)

}
```

```
helloWorld()
```

```
## Hello World!
```

Function: Hello… you!

```
helloWorld2 <- function(to="World") {

  outmessage <- paste("Hello", to, "!\n", sep=" ")

  cat(outmessage)

}
```

Function: Hello… you!

```
helloWorld2 <- function(to="World") {

  outmessage <- paste("Hello", to, "!\n", sep=" ")

  cat(outmessage)

}
```

```
helloWorld2()
```

```
## Hello World !
```

```
helloWorld2("Francesca")
```

```
## Hello Francesca !
```

Function: BMI

Let's build a function that computes the body mass index (BMI) taking as arguments the weigth in kg and the height in meters.

```
BMI <- function (weight, height) {

  bmi <- weight/height^2

  return(bmi)

}
```

Note: the return function returns the final result/object produced by the function and stops any other evaluation. Any code written after the return statement is not evaluated.

Function: BMI

Now we can calculate the BMI of an individual who weights 80kg and is 1.90m tall

```
BMI(80, 1.90)
```

```
## [1] 22.16066
```

Respect argument order or specify their names

```
BMI(1.90, 80)
```

```
## [1] 0.000296875
```

```
BMI(height=1.90, weight=80)
```

```
## [1] 22.16066
```

Exercises

Ex. 2.7

```
x <- c(1, 2, 200, 6, 80, 23)
y <- c(100, 50, 5, 30, 1, 20)
```

Compute:

- Mean of *x* and *y*
- Variance of *x* and *y*
- Spearman's correlation of *x* vs. *y*

Ex. 2.8

```
mLen <- c(31, 28, 31, 30,
  31, 30, 31, 31,
  30, 31, 30, 31)

names(mLen) <- c("Jan", "Feb", "Mar", "Apr",
  "May", "Jun", "Jul", "Aug",
  "Sep", "Oct", "Nov", "Dec")
```

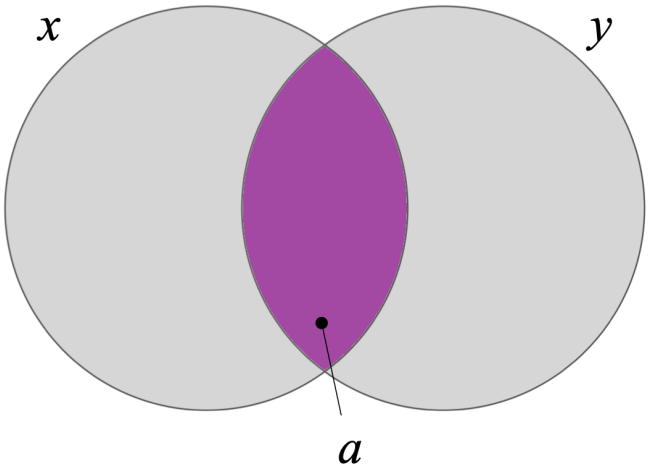
Use the table function to know how many months are 31 days long

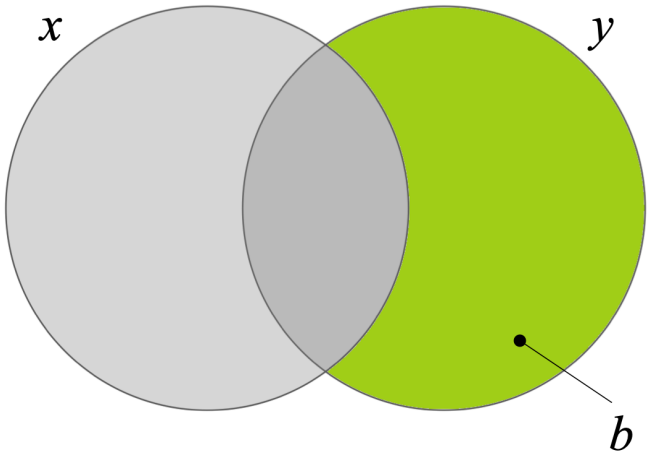
Ex. 2.9

Use the sample function to create a vector *x* containing the first 10 numbers obtained in a bingo extraction (possible values from 1 to 90).

Sort the numbers and save them in vector *y*.

Ex. 2.10

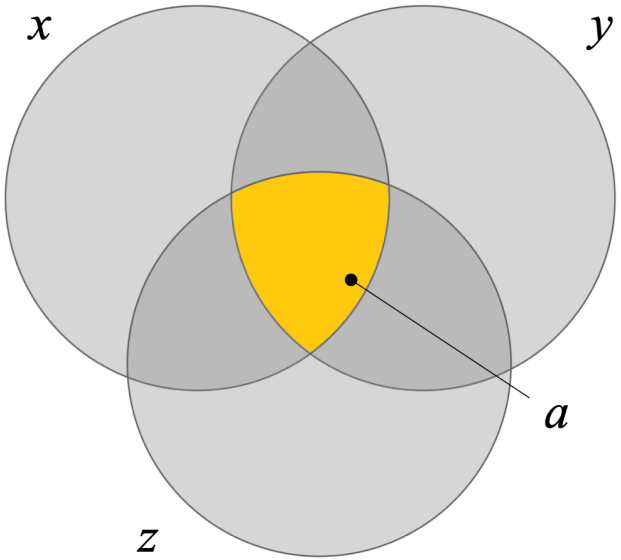


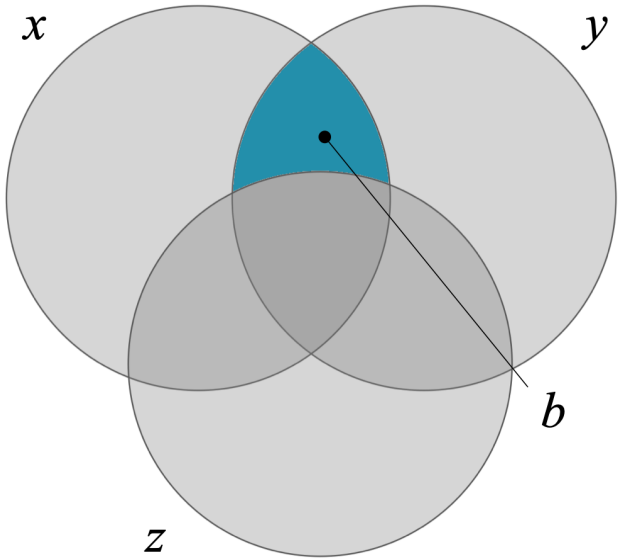



```
x <- seq(1,5)
y <- seq(3,7)
```

Use intersect and setdiff to identify the elements belonging to the coloured areas in the figure above. Save them into two variables called *a* and *b*

Ex. 2.11






```
x <- c(2, 4, 5, 7, 8)
y <- c(2, 4, 6, 8, 10)
z <- c(2, 3, 4, 5, 6)
```

Identify the elements belonging to the coloured areas and save them into two variables *a* and *b*. Hint: intersect and setdiff only accept two sets, but can be combined recursively (e.g. intersection of the intersection).

46/52

Ex. 2.12

Create a function called firstNsum that for a positive integer *n* selected by the user, computes the sum of the first *n* positive numbers (Hint: use seq and sum).

Verify for n=10, 20, and 100 that this sum is equal to $\frac{n(n+1)}{2}$

Ex. 2.13

Create a function called FtoCtemp that takes as input the temperature in Fahrenheits and converts it into Celsius using the following formula:

$$C = \frac{5}{9}(F - 32)$$

Use this function to determine wheter your are going to sleep confortably in your hotel room in New York City, where the temperature is set to 47°F.

Ex. 2.14, optional

Create a function called top3 that takes as input a vector of numbers, sorts it, and extracts the top three highest values (Hint: use sort and tail).

Note: it should also work for vectors with less than three elements.

Ex. 2.15

Create a function called bestFriends that considers four input parameters:

- name (character) and year of birth (numeric) of the 1st person
- name and year of birth of the 2nd person

and computes the probability (between 0% and 100%) that they will become best friends.

Compute the probability as you wish (be creative!), but the function should give the same result when the same input names and years are used and tnot exceed the 0-100% interval.

Test with the colleague at your right side what is your probability of becoming best friends according to your two functions.

Useful resources

R for beginners: https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

Basic R functions: http://www.sr.bham.ac.uk/~ajrs/R/r-function_list.html

Solutions

Ex. 2.3: 3) 5

Ex. 2.6: 3) Factor

Ex. 2.7: mean_x=52; mean_y=34.33; cor_xy=-0.94

Ex. 2.7: 7

Ex. 2.13: 8.33°C