

R: BASICS

Andrea Passarella

(plus some additions by Salvatore Ruggieri)



BASIC CONCEPTS

- R is an interpreted scripting language
- Types of interactions
 - **Console** based
 - Input commands into the console
 - Examine results
 - **Scripting**
 - Sequence of statements in a text file
 - Use the "source()" command to process the file
 - Equivalent to provide the sequence of statements to the console
- How we will use it
 - **Variables** to store data
 - **Functions** (either existing in the packages or new ones written on purpose) to process data
 - (Limited) **I/O with external files** for
 - Input/output of data
 - scripting

```
Console ~/R_working_dir/ ↵

R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/R_working_dir/.RData]

Loading required package: Matrix
> |
```

LAUNCH, HELP, SAVE, EXIT



- Launching the “R” application means running the interpreter shell

```
18:39 andrea R $ R

R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

LAUNCH, HELP, SAVE, EXIT

- RStudio is a front-end to the language
 - Embeds the interpreter shell (Console)
 - Visualisation of available variables
 - Package installation
 - Help

The screenshot shows the RStudio interface with the following components:

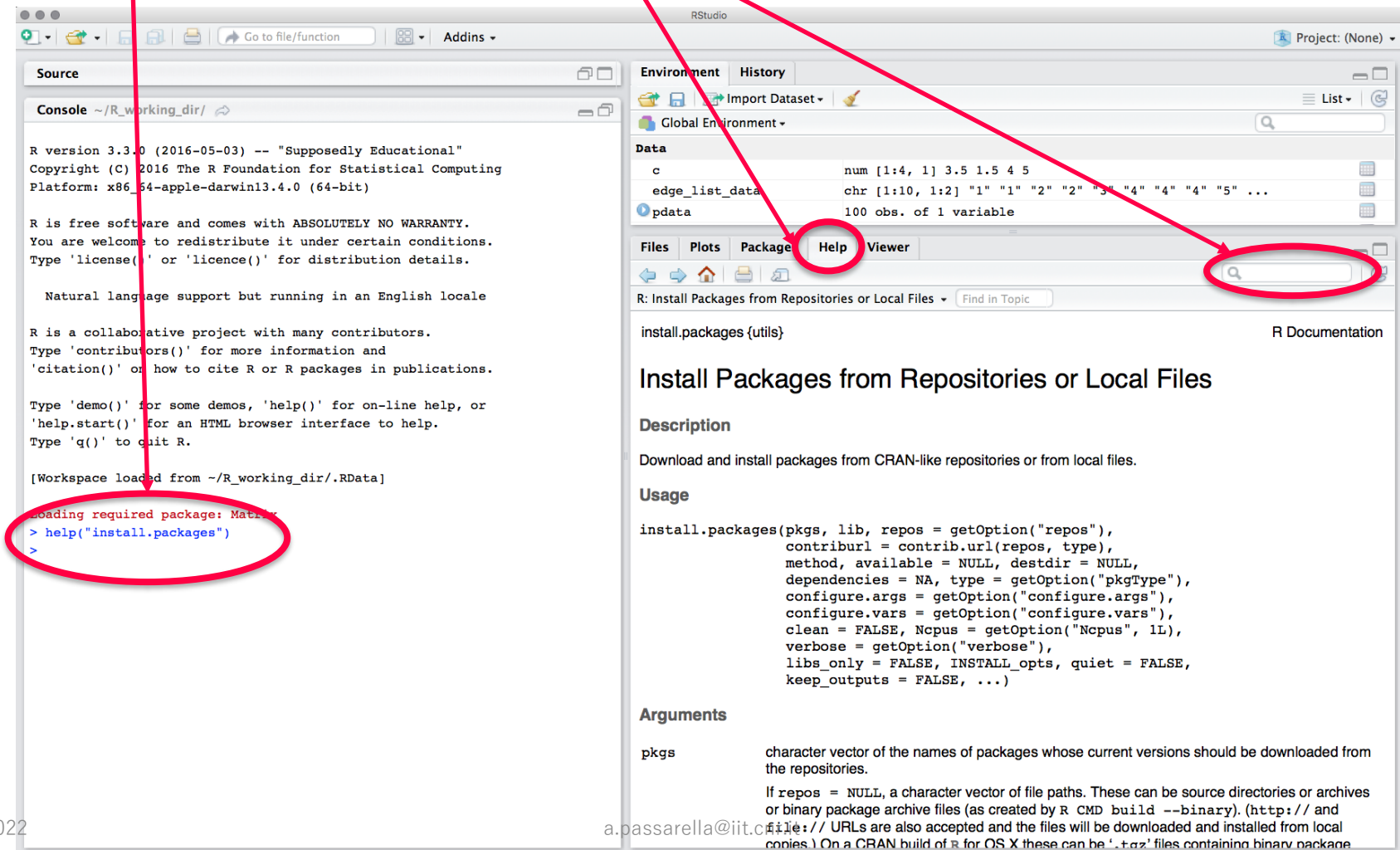
- Source Pane:** Shows the R version 3.3.0 (2016-05-03) and the R Foundation copyright notice.
- Console:** Displays the R startup message, including the license information and the command prompt. The output shows the workspace loaded from the current directory and the loading of the Matrix package.
- Environment Pane:** Shows the current environment with variables: 'c' (numeric vector), 'edge_list_data' (character vector), and 'pdata' (data frame).
- Files Pane:** Shows the file explorer with options to install or update packages.
- Plots Pane:** Shows the plot area.
- Packages Pane:** Shows a list of installed and available packages, including the System Library and various Bioconductor packages.

Red arrows from the text above point to the Console, Environment, and Packages panes.

LAUNCH, HELP, SAVE, EXIT

- Help

- Search with the **user interface**
- **help()** function from the **console**



The screenshot shows the RStudio interface. On the left, the console displays the R version information and the execution of the `help("install.packages")` command. A red circle highlights the command in the console. On the right, the R Documentation pane shows the help page for `install.packages`. A red circle highlights the search bar in the top right of the documentation pane, and a red arrow points from the search bar to the console command. Another red circle highlights the 'Help' button in the top navigation bar.

```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/R_working_dir/.RData]
loading required package: Matrix
> help("install.packages")
>
```

install.packages {utils} R Documentation

Install Packages from Repositories or Local Files

Description

Download and install packages from CRAN-like repositories or from local files.

Usage

```
install.packages(pkgs, lib, repos = getOption("repos"),
  contriburl = contrib.url(repos, type),
  method, available = NULL, destdir = NULL,
  dependencies = NA, type = getOption("pkgType"),
  configure.args = getOption("configure.args"),
  configure.vars = getOption("configure.vars"),
  clean = FALSE, Ncpus = getOption("Ncpus", 1L),
  verbose = getOption("verbose"),
  libs_only = FALSE, INSTALL_opts, quiet = FALSE,
  keep_outputs = FALSE, ...)
```

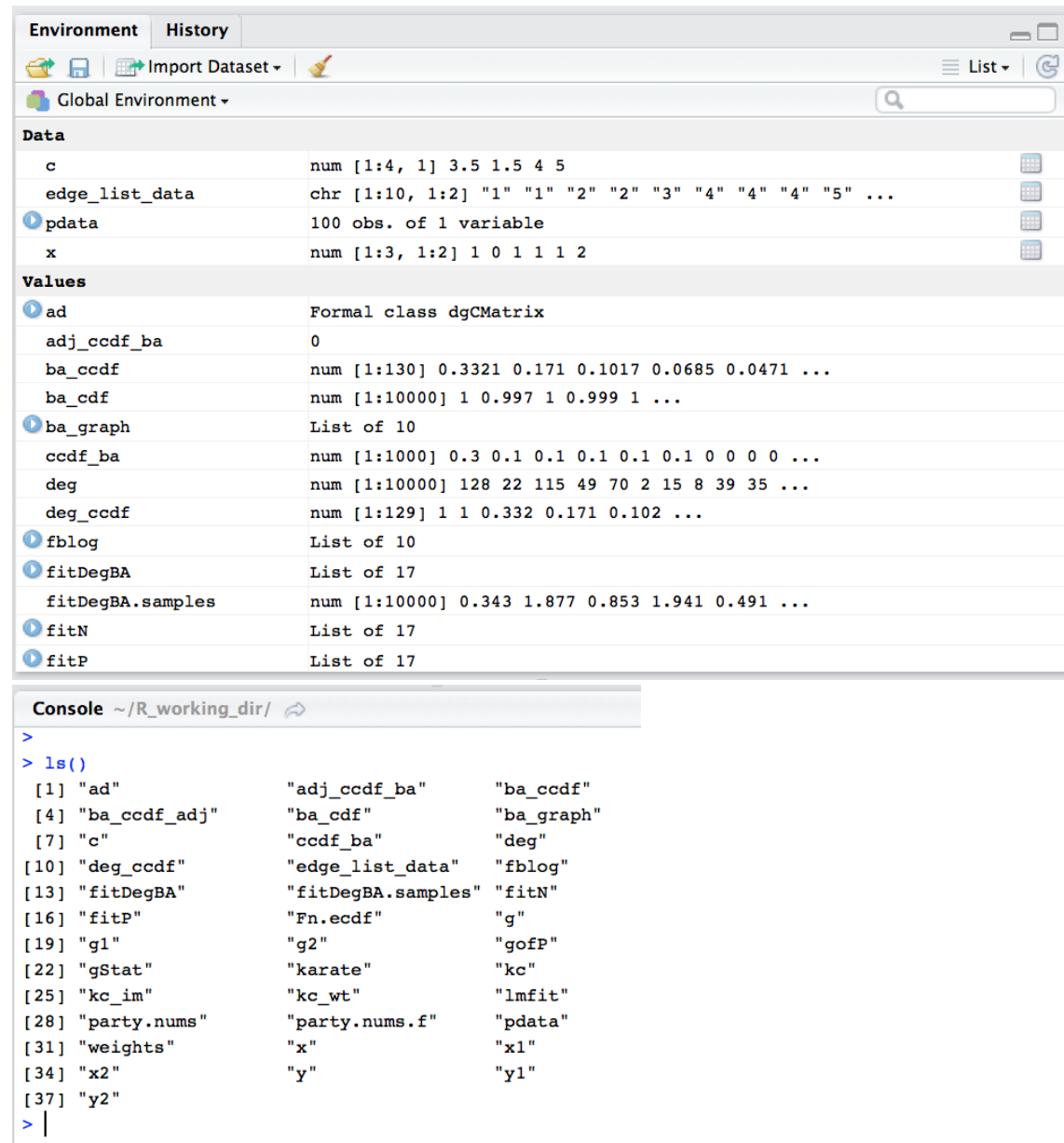
Arguments

pkgs character vector of the names of packages whose current versions should be downloaded from the repositories.

If `repos = NULL`, a character vector of file paths. These can be source directories or archives or binary package archive files (as created by R CMD `build --binary`). (`http://` and `file://` URLs are also accepted and the files will be downloaded and installed from local copies.) On a CRAN build of R for OS X these can be `.tar.gz` files containing binary package

LAUNCH, HELP, SAVE, EXIT

- **Workspace** = set of data, function, ... defined during a session
- The elements of the workspace are shown in the “**Environment**” pane
- or can be listed with `ls()` from the console



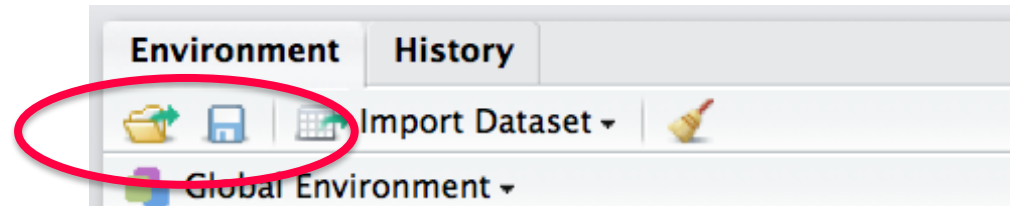
The screenshot shows the RStudio Environment pane with the following variables and their types:

Variable	Type
c	num [1:4, 1] 3.5 1.5 4 5
edge_list_data	chr [1:10, 1:2] "1" "1" "2" "2" "3" "4" "4" "4" "5" ...
pdata	100 obs. of 1 variable
x	num [1:3, 1:2] 1 0 1 1 1 2
Values	
ad	Formal class dgCMatrix
adj_ccdf_ba	0
ba_ccdf	num [1:130] 0.3321 0.171 0.1017 0.0685 0.0471 ...
ba_cdf	num [1:10000] 1 0.997 1 0.999 1 ...
ba_graph	List of 10
ccdf_ba	num [1:1000] 0.3 0.1 0.1 0.1 0.1 0.1 0 0 0 0 ...
deg	num [1:10000] 128 22 115 49 70 2 15 8 39 35 ...
deg_ccdf	num [1:129] 1 1 0.332 0.171 0.102 ...
fblog	List of 10
fitDegBA	List of 17
fitDegBA.samples	num [1:10000] 0.343 1.877 0.853 1.941 0.491 ...
fitN	List of 17
fitP	List of 17

The Console shows the output of the `ls()` command:

```
> ls()
 [1] "ad"                "adj_ccdf_ba"       "ba_ccdf"
 [4] "ba_ccdf_adj"      "ba_cdf"            "ba_graph"
 [7] "c"                 "ccdf_ba"           "deg"
[10] "deg_ccdf"         "edge_list_data"    "fblog"
[13] "fitDegBA"         "fitDegBA.samples" "fitN"
[16] "fitP"             "fn.ecdf"           "g"
[19] "g1"               "g2"                "gofP"
[22] "gStat"           "karate"            "kc"
[25] "kc_im"           "kc_wt"             "lmfit"
[28] "party.nums"      "party.nums.f"      "pdata"
[31] "weights"         "x"                 "x1"
[34] "x2"              "y"                 "y1"
[37] "y2"
> |
```

- Workspaces can be **saved** and **restored** from previous sessions
 - Either through the UI in RStudio



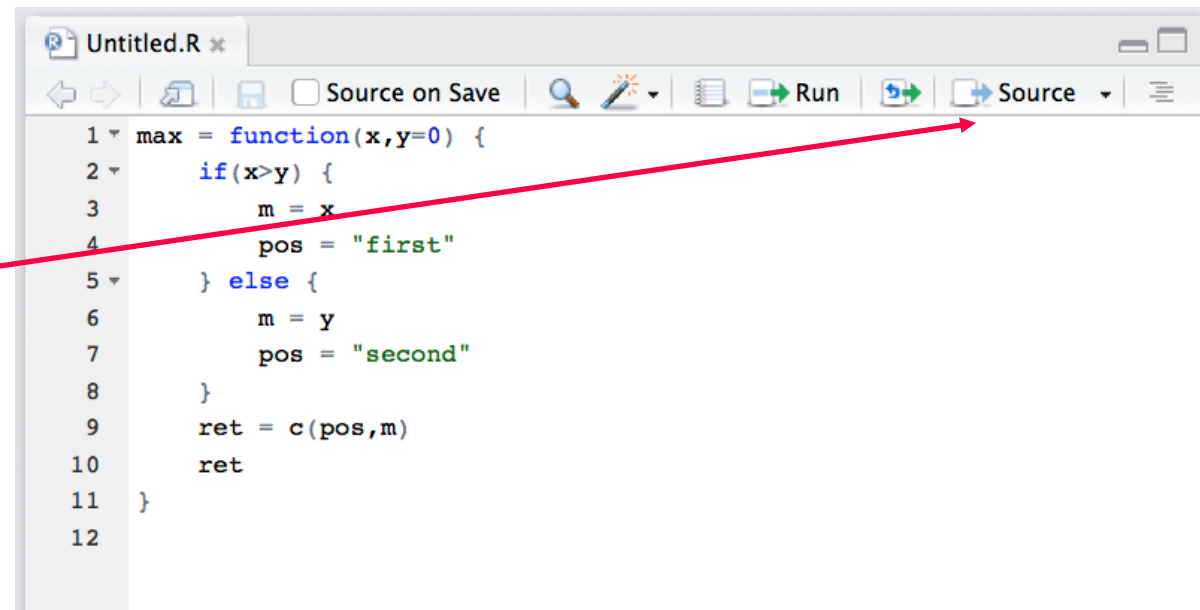
- or via `save.image()` and `load()` functions from the R console

```
>  
> load("~/R_working_dir/Untitled.RData")  
> save.image("~/R_working_dir/Untitled.RData")  
>
```

- **Automatic actions** (upon running/exiting from R/RStudio)
 - Load workspace from a file ".RData" in the working directory upon launch
 - Ask to save to ".RData" in the working directory upon exiting

SCRIPTING

- For non-toys use, most likely you want to
 - Write a script with a set of R statements
 - Execute the script and get the results
- Writing a script
 - Write the script as a text file in any text editor
 - NOT using Word, using a real file text editor
 - Use the file editor integrated in RStudio
- Execute the script
 - Using the `source()` function
 - Loading the script file into the editor and “sourcing” from there



The screenshot shows the RStudio editor window titled 'Untitled.R'. The code in the editor is as follows:

```
1 max = function(x,y=0) {  
2   if(x>y) {  
3     m = x  
4     pos = "first"  
5   } else {  
6     m = y  
7     pos = "second"  
8   }  
9   ret = c(pos,m)  
10  ret  
11 }  
12
```

The toolbar at the top of the editor includes buttons for 'Run' and 'Source'. A red arrow points from the text '“sourcing” from there' in the list above to the 'Source' button in the toolbar.

- Function `data()` list the set of available dataset provided by the currently loaded packages
- `data(iris)` loads data from iris (the name of the dataset) in the current workspace
 - a variable (a `dataframe`, see later) called `iris` is added to the workspace
- Depending on the dataset format, it might be needed to access the dataframe to “expand” it
 - E.g., `ls(iris)`

VARIABLES

- Defined as they are needed
- Assignment operator, `<-`, or `=`
 - `a = 15` defines variable `a`, with value 15
 - From then on, `a` becomes available in the workspace

- Looking into variables
 - Type the name in the console

```
> a = 15
> a
[1] 15
> |
```

- `summary(variable_name)` shows a summary, which depends on the type of the variable
 - e.g., if `p` is a set of values, `summary(p)` shows some reference percentiles of these values

```
> p
[1] 0.4970180 0.2177386 0.1030616 0.4776593 0.7038415 0.9508472
[7] 0.3151198 0.7208926 0.4440492 0.6947185
> summary(p)
  Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
0.1031  0.3474  0.4873  0.5125  0.7016  0.9508
>
```

VECTORS, ARRAYS

- Vectors are the most basic structure in R
 - a collection of values of the same type

Function `c()`, returns a **collection** of the arguments

```
> a = c(1,5,10)
```

```
> a
```

```
[1] 1 5 10
```

```
> b = c("mela", "pera", "albicocca", 5)
```

```
> b
```

```
[1] "mela"      "pera"      "albicocca" "5"
```

```
> |
```

`a` is a vector of integers

`b` is a vector of **character strings**

- note the **difference** between
 - 5 in a
 - "5" in b

VECTORS, ARRAYS

- Arrays are **vectors** with given **dimensions**

```
> a = c(1,2,3,4,5,6,7,8,9,10)
> a
 [1] 1 2 3 4 5 6 7 8 9 10
> dim(a)
NULL
> dim(a)=10
> dim(a)
 [1] 10
> a
 [1] 1 2 3 4 5 6 7 8 9 10
> dim(a)=c(2,5)
> a
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Collection of values without any specific dimension attribute

Now gets a single dimension

2 dimensions, 2 rows, 5 columns

VECTORS, ARRAYS

- Arrays can be created more simply with `array()`

```

> a
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> b = array(seq(1:10), dim=c(2,5))
> b
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

```

`seq()` generates a sequence of values between the given extremes

`dim` parameter of the function to set the dimensions

- Matrices** are arrays with 2 dimensions only

- Note that arrays can have more than 2 dimensions

```

> c = matrix(seq(1:10), nrow = 2, ncol = 5)
> c
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

```

ACCESSING VECTOR/ARRAY ELEMENTS

- The `[]` operator
 - Start counting from 1, not from 0!

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

```
> c[1,3] ← Element with index (1,3)
[1] 5
> c[1,] ← All elements of the first row
[1] 1 3 5 7 9
> c[,2] ← All elements of the second column
[1] 3 4
```

NB: `c[,2]` is itself a vector, thus one can further index it

```
> c[,2][1] ← First element of c[,2]
[1] 3      (equivalent to c[1,2])
```

ACCESSING VECTOR/ARRAY ELEMENTS

- **Negative** indices

- `c[, -2]`: c with all columns but 2
- In general, negative indices are excluded, e.g. `c[, c(-1; -3)]`

variable `c` combination function `c()`



- **Range** indices

- `c[, 2:4]`: all columns of matrix c between 2 and 4

- Expressions as indices

- `c[c>5]`: all values greater than 5
- `c[c>5 & c<10]`: all values between 5 and 10
 - return value is a vector

```
> c[, -2]
      [,1] [,2] [,3] [,4]
[1,]    1    5    7    9
[2,]    2    6    8   10
> c[, c(-1, -3)]
      [,1] [,2] [,3]
[1,]    3    7    9
[2,]    4    8   10
>
```

```
> c[, 2:4]
      [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
```

```
> c[c>5]
[1]  6  7  8  9 10
```

```
> c[c>5 & c<10]
[1] 6 7 8 9
```

- Standard set of operators of any programming language
 - ! Unary not
 - < Less than, binary
 - > Greater than, binary
 - == Equal to, binary
 - >= Greater than or equal to, binary
 - <= Less than or equal to, binary
 - & And, binary, **vectorized**
 - && And, binary, not vectorized
 - | Or, binary, **vectorized**
 - || Or, binary, not vectorized

LOGICAL OPERATORS: VECTORISED VS NON-VECTORISED



- `c[c>5 & c<10]`: all values between 5 and 10

- Steps

- `c>5`: a matrix of the same dimensions of `c`, with TRUE or FALSE values

- `c<10`

- `c>5 & c<10`: a matrix of the same dimensions of `c`, with the logical AND of the two expressions

- `c[c>5 & c<10]`: select from `c` only the elements for which the indices are TRUE

```
> c[c>5 & c<10]
[1] 6 7 8 9
```

```
> c>5
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE TRUE TRUE TRUE
```

```
> c<10
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,] TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE FALSE
```

```
> c>5 & c<10
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE TRUE TRUE FALSE
```

```
> c
```

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

LOGICAL OPERATORS: VECTORISED VS NON-VECTORISED



- `c[c>5 & c<10]`: all values between 5 and 10
- Steps
 - `c>5`: a matrix of the same dimensions of `c`, with TRUE or FALSE values
 - `c<10`
- `c>5 & c<10`: a matrix of the same dimensions of `c`, with the logical AND of the two expressions
 - We need to do the logical AND on an element-by-element of the two matrices
 - This is obtained with the **vectorised** version of the operator, “&”
- `c>5 && c<10`: non-vectorised version
 - Applicable to single-element data
 - In case of vectors stops at the first element
 - Typically used for indices in **control statements** and **loops**

```
> c[c>5 & c<10]
[1] 6 7 8 9
```

```
> c>5
      [,1] [,2] [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE TRUE TRUE TRUE
```

```
> c<10
      [,1] [,2] [,3] [,4] [,5]
[1,] TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE FALSE
```

```
> c>5 & c<10
      [,1] [,2] [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE TRUE TRUE FALSE
```

```
> c>5 && c<10
[1] FALSE
```

BUILDING MATRICES

- Sometimes useful to build matrices by **stitching together** existing arrays or matrices
 - `cbind()` joins together vectors/matrices by column
 - `rbind()` joins together vectors/matrices by row

```
> d = c(11,12)
> c = cbind(c,d,deparse.level = 0)
> c
```

Do not assign names to columns

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	3	5	7	9	11
[2,]	2	4	6	8	10	12

previous matrix c

vector d

LISTS, DATA FRAMES

- Lists are collections of **arbitrary** data types

```
> Lst <- list(name="Fred", wife="Mary",  
+           no.children=3, child.ages=c(4,7,9))
```

```
> Lst
```

```
$name
```

```
[1] "Fred"
```

character string

```
$wife
```

```
[1] "Mary"
```

```
$no.children
```

```
[1] 3
```

integer

```
$child.ages
```

```
[1] 4 7 9
```

vector of 3 elements

```
> length(Lst$name)
```

```
[1] 1
```

```
> length(Lst$wife)
```

```
[1] 1
```

```
> length(Lst$no.children)
```

```
[1] 1
```

```
> length(Lst$child.ages)
```

```
[1] 3
```

Function `length()`

- size of the variable
- different from `dim()`

```
> d = c(11,12)
```

```
> length(d)
```

```
[1] 2
```

```
> dim(d)
```

```
NULL
```

LISTS, DATA FRAMES

- Data frames
 - **lists** whose **components** are all of the **same length**
 - If components are seen as columns of a matrix, all columns must have the same size
 - With respect to **matrices**, columns can be of **different types**

```
> name = c("Pietro", "Paolo", "Antonio")
> city = c("Pisa", "Pisa", "Ancona")
> age = c(25, 15, 34)
> df = data.frame(name,city,age)
> df
```

	name	city	age
1	Pietro	Pisa	25
2	Paolo	Pisa	15
3	Antonio	Ancona	34

Note the difference with the definition of `Lst`!

```
> df = data.frame(name="Fred", wife="Mary",
+                 no.children=3, child.ages=c(4,7,9))
> df
```

	name	wife	no.children	child.ages
1	Fred	Mary	3	4
2	Fred	Mary	3	7
3	Fred	Mary	3	9

ACCESSING ELEMENTS OF LISTS AND DATAFRAMES



- `$` or `[[]]` operator
 - Selection of elements in a list or data frame
 - Either by position: `df[[1]]`
 - Or by name: `df[["name"]]`, `df$name`

- Levels are the **unique elements** found, if defined

```
> df[[1]]
[1] Pietro Paolo Antonio
Levels: Antonio Paolo Pietro
> df[["name"]]
[1] Pietro Paolo Antonio
Levels: Antonio Paolo Pietro
> df$name
[1] Pietro Paolo Antonio
Levels: Antonio Paolo Pietro
```

```
> df$city
[1] Pisa Pisa Ancona
Levels: Ancona Pisa
```

ADDING REMOVING ELEMENTS FROM LISTS/DATA FRAMES



- Assigning `NULL` to an element drops that element


```
> df
  name  city age
1 Pietro  Pisa 25
2 Paolo  Pisa 15
3 Antonio Ancona 34
```



```
> df$age = NULL
> df
  name  city
1 Pietro  Pisa
2 Paolo  Pisa
3 Antonio Ancona
```

- Create a new element by just assigning values to the name of the new element

```
> df$age = NULL
> df
  name  city
1 Pietro  Pisa
2 Paolo  Pisa
3 Antonio Ancona
```



```
> age
[1] 25 15 34
> df$age = age
> df
  name  city age
1 Pietro  Pisa 25
2 Paolo  Pisa 15
3 Antonio Ancona 34
```

MODIFYING ELEMENTS IN A LIST/DATA FRAME

- `[[]]` or `$` operators return a **vector**
 - Whose elements can be managed with the normal index operators
 - E.g., `[]`

```
> df
```

	name	city	age
1	Pietro	Pisa	25
2	Paolo	Pisa	15
3	Antonio	Ancona	34

```
> df$age[1]
```

```
[1] 25
```

```
> df$age[1] = 10
```

```
> df
```

	name	city	age
1	Pietro	Pisa	10
2	Paolo	Pisa	15
3	Antonio	Ancona	34

DATA FRAMES AS MATRICES

- Sometimes it is useful to access Data Frames as matrices

Names of the **rows**

- Access and modify via `rownames(df)`

```
> df
```

	name	city	age
1	Pietro	Pisa	10
2	Paolo	Pisa	16
3	Antonio	Ancona	35

Names of the **columns**

- Access and modify via `colnames(df)`

Matrix part of the data frame

Access and modify via the `[,]` operator

```
> df[,1]
[1] Pietro Paolo Antonio
Levels: Antonio Paolo Pietro
```

```
> df[2,]
  name city age
2 Paolo Pisa  16
```

```
> df[1,3] = 25
```

```
> df
  name    city age
1 Pietro  Pisa  25
2 Paolo  Pisa  16
3 Antonio Ancona 35
```

Select people whose age is greater than 16

```
> temp = df$age>16
> temp
[1] TRUE FALSE TRUE
> df[temp,]
```

```
  name    city age
1 Pietro  Pisa  25
3 Antonio Ancona 35
```

T/F index vectors can also be applied to columns!

- Select only those columns for which the condition is true

```
> df[,temp]
  name age
1 Pietro 25
2 Paolo 16
3 Antonio 35
```

ARITHMETIC OPERATIONS

- With arrays, **element-by-element** operation

```

> a = array(c(1,2,3))      > prod = a*b
> a                        > prod
[1] 1 2 3                  [1] 8 18 30
> b = array(c(8,9,10))
> b
[1] 8 9 10

```

- Same semantic with matrices

```

> c                        > c*d
      [,1] [,2] [,3] [,4] [,5]      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9      [1,]    1    9   50   49   81
[2,]    2    4    6    8   10      [2,]    4   16   36   64  100
> d
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3   10    7    9
[2,]    2    4    6    8   10

```

- Use “**%*%**” for the standard matrix **product form**

```

> c                        > d                        > c %*% d
      [,1] [,2] [,3] [,4] [,5]      [,1] [,2]      [,1] [,2]
[1,]    1    3    5    7    9      [1,]    1    6      [1,]   140   220
[2,]    2    4    6    8   10      [2,]    2    7      [2,]   160   260
                                   [3,]    3    8
                                   [4,]    4    9
                                   [5,]   10   10

```

CONDITIONAL STATEMENT

- General form
 - If (statement1)
 statement2
 else
 statement3
- Example
 - if (x > 0) {
 count = count+1
 x = x+1
 print(x)
} else {
 count = count-1
 x = x-1
 print(x)
}

```
> x = 5
> count = 1
> if (x > 0) {
+     count = count + 1
+     x = x + 1
+     print(x)
+ } else {
+     count = count - 1
+     x = x-1
+     print(x)
+ }
[1] 6
```

LOOP STATEMENT

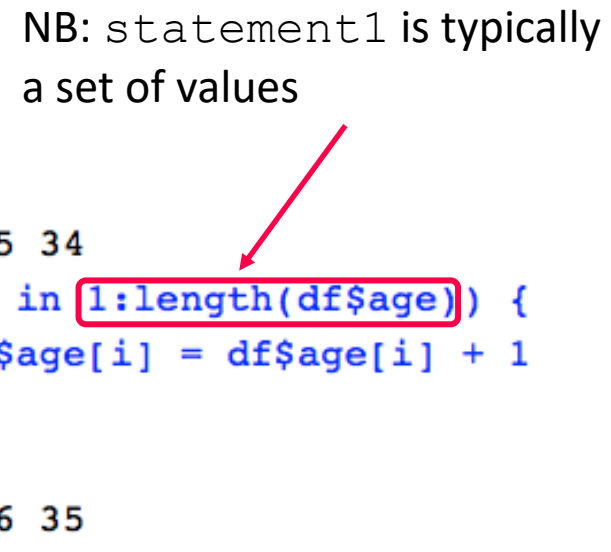
- **While** loop
 - while (expression)
statement

```
> x = 5
> count = 2
> while (count > 0) {
+     x = x-1
+     count = count-1
+ }
> print(x)
[1] 3
```

- **For** loop
 - for (name in statement1) statement2

NB: statement1 is typically
a set of values

```
> df$age
[1] 25 15 34
> for (i in 1:length(df$age)) {
+     df$age[i] = df$age[i] + 1
+ }
> df$age
[1] 26 16 35
```



FUNCTIONS

- General form
 - `name <- function(arg_1, arg_2, ...)`
expression
- Return the max of two arguments

```
> max = function(x,y) {  
+   if(x>y)  
+     ret = x  
+   else  
+     ret = y  
+   ret  
+ }  
> max(3,5)  
[1] 5
```

- Return the max and whether it was first or second argument

```
> max = function(x,y) {  
+   if(x>y) {  
+     m = x  
+     pos = "first"  
+   } else {  
+     m = y  
+     pos = "second"  
+   }  
+   ret = c(pos,m)  
+   ret  
+ }
```

```
> max(3,5)  
[1] "second" "5"  
> max(5,3)  
[1] "first" "5"
```

DEFAULT AND NAMED ARGUMENTS

- Functions may be defined with default arguments

```
> max = function(x, y=0) {
+   if(x>y) {
+     m = x
+     pos = "first"
+   } else {
+     m = y
+     pos = "second"
+   }
+   ret = c(pos,m)
+   ret
+ }
```

```
> max(5)
[1] "first" "5"
> max(-10)
[1] "second" "0"
> max(-10,-20)
[1] "first" "-10"
```

- Parameters can also be given by name (instead of by position)

```
> max(-10,-20)
[1] "first" "-10"
> max(y=-10,x=-20)
[1] "second" "-10"
```

IMPLICIT LOOPS

- `lapply(ls, f)`
 - Applies function `f()` to each element of list `ls`. Returns a list of results.
- `sapply(ls, f)`
 - Applies function `f()` to each element of list `ls`. Returns an array of results.

```
> grades = list(dsd=c(28, 30, 26), dm=c(25, 25, 28, 30))
> grades
$dsd
[1] 28 30 26

$dm
[1] 25 25 28 30

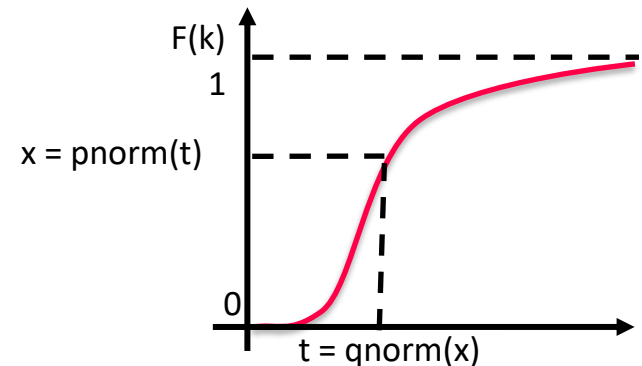
> lapply(grades, mean)
$dsd
[1] 28

$dm
[1] 27

> sapply(grades, mean)
 dsd  dm
 28   27
```

PROBABILITY DISTRIBUTIONS

- R includes a **family of functions** to manage the most popular distributions
- Given a specific distribution (e.g., normal, named “norm” in R)
 - `rnorm(100, mean=0, std=1)`
 - Generates 100 **samples** from a normal distribution with mean 0 and standard deviation 1
 - `dnorm(3, mean=0, std=1)`
 - **Density** function computed at 3 ($f(3)$)
 - `pnorm(3, mean=0, std=1)`
 - **Distribution** function computed at 3 ($F(3) = P(X \leq 3) = 0.9986501$)
 - `qnorm(0.9986501, mean=0, std=1)`
 - **Percentile** corresponding to 0.9986501 (t s.t. $P(X \leq t) = 0.9986501$)
- Given a set of values in a vector x
 - `mean(x)` gives the average
 - `sd(x)` gives the standard deviation
 - `Summary(x)` gives a summary of the main percentiles of the distribution



PROBABILITY DISTRIBUTIONS

- Parameters to the p,q,r,d functions depend on the particular distribution

- See also

Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logistic	logis	location, scale
negative binomial	nbinom	size, prob
normal	norm	mean, sd
Poisson	pois	lambda
signed rank	signrank	n
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

BASIC I/O

- Read values into a vector
 - `scan()` function

File "sample.txt"

```
FIRST LINE
1 10 20
30 40 50
```

```
> l = scan("sample.txt", skip=1)
Read 6 items
> l
[1] 1 10 20 30 40 50
```

Initial lines to skip

A **path** to the file to read

- if relative, the working directory is assumed
- Use `getwd()` for the name of the working directory
- Equivalent to `paste(getwd(), "/sample.txt", sep="")`

By default, elements are separated by **white spaces** or end-of-line

- can be modified through the **sep** argument


BASIC I/O

- Read structured data into data frames
 - `read.table()` function

File "sample.txt"

```
first second third
1      10      20
30     40     50
```

```
> df_read = read.table("sample.txt",
+                       header = T)
> df_read
  first second third
1      1     10    20
2     30     40    50
```



Whether the first line should be used to get the column names

WRITING DATA FRAMES TO FILES

- `write.table()` function

Data frame to write

Where to write it

```
> write.table(df_read, "out_df.txt",  
+             row.names = F, quote=F, sep='\t')
```

Whether to put row names
(usually numbers) in front of rows

Use tab as separator

Whether to put quotes around character strings

File "out_df.txt"

```
first    second    third  
1         10        20  
30        40        50
```

WRITING VECTORS, LISTS, OR MATRICES

- `write()` function

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

Object to write

Where to write it

```
> write(c, "out_matrix.txt",
+       sep='\t', ncolumns = dim(c)[2])
```

Use tab as separator

Number of columns in the output file

- Here equal to the number of columns of the matrix
- Same with function `ncol(c)`

File "out_matrix.txt"

```
1      2      3      4      5
6      7      8      9     10
```

1. Install (if needed) the MASS package and load it
2. Load the “Animals” data set
3. Calculate the ratio between animals' brain size and their body size, adding the result as a new column called “`proportions`” to the Animals data frame
4. Calculate average and standard deviation of the “`proportions`”
5. Remove the column “`proportions`” from the data frame
6. Select animals with body size > 100
7. Get a list of animals' names with body size > 100 and brain size > 100

8. Find the average body and brain size for the first 10 animals in the dataset
9. Write a function that returns a list of two elements containing the mean value and the standard deviation of a vector of elements
 - Apply this to the body and brain sizes of Animals
10. Create a vector called `body_norm` with 100 samples from a Normal random variable with average and standard deviation equal to those of body sizes in the Animals dataset
 - print the summary of the generated dataset
 - compare the summary with another dataset of 100 samples with same average and $sd = 1$
11. Save the Animals data frame to a file named “`animals_a.txt`” with row and column names
12. Create a copy of the file named “`animals_b.txt`”, then
 - modify some data in it
 - Read the file into a new data frame, `Animals_b`
 - Write a function that returns the rows that differ between `Animals` and `Animals_b`
13. Save the workspace to a file, clean the workspace, restore the workspace from the file