Programming and frameworks for ML

Introduction to R

1

Presentation

Big Data Consultant at Indra / Big Data Lecturer

- More than 20 years of experience in different environments, technologies, customers, countries ...
- Passionate data and technology
- Enthusiastic Big Data world and NoSQL

Daniel Villanueva Jiménez

Consultor BigData en Indra

NDRA • Universidad Pontificia de Salamanca

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

What is R?

- R is a programming language and environment with the aim of displaying information, statistics and analysis computations.
- R is an open implementation source S programming language, developed by Bell Laboratories in 1976
- R has a very active user community
  - More than 10,000 packages CRAN

Pros and cons

- In favor:
  - Free
  - Many packages, very flexible.
  - It can run on virtually any combination of hardware / operating system (even in a PlayStation 3)

- Against:
  - The objects are to be stored in the physical memory of the computer
  - Much more oriented programming
  - Minimum interface

Functionalities in R

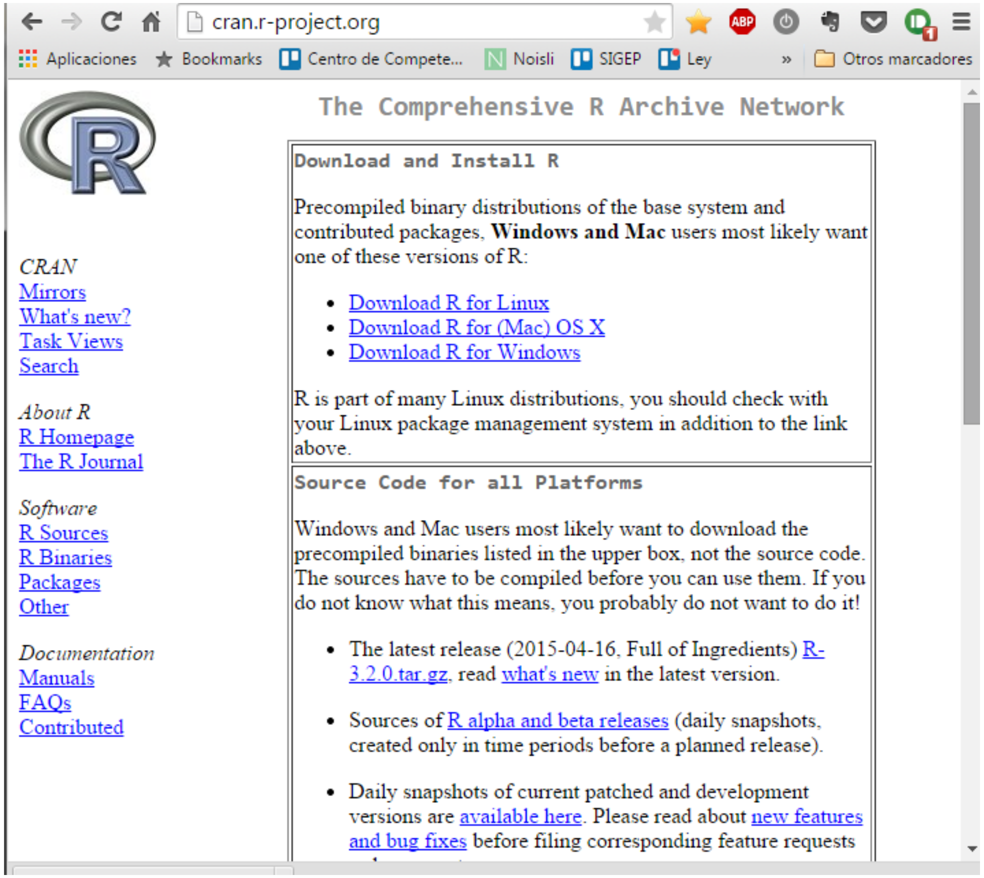
- Classification and Regression
- Mining and text analysis
- Models of scoring and ranking
- Clustering
- Graph analysis
- Interactive data analysis
- Data handling and cleaning
- Data Visualization
- General purpose statistics
- Advanced predictive models (SVM, NN, ...)

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Exercise 1

- Install R (http: //cran.r-project.org/)



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

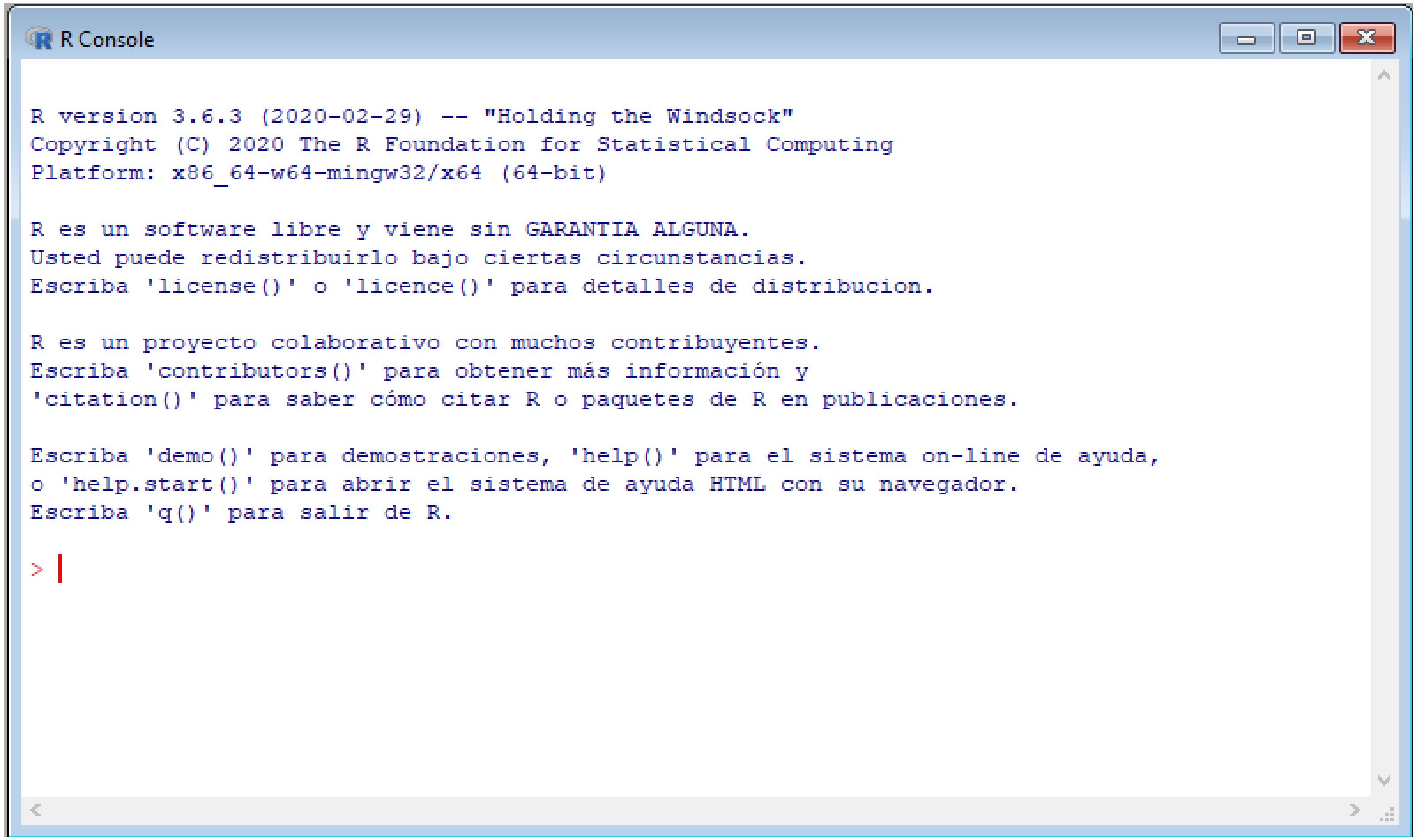
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2015-04-16, Full of Ingredients) R-3.2.0.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests

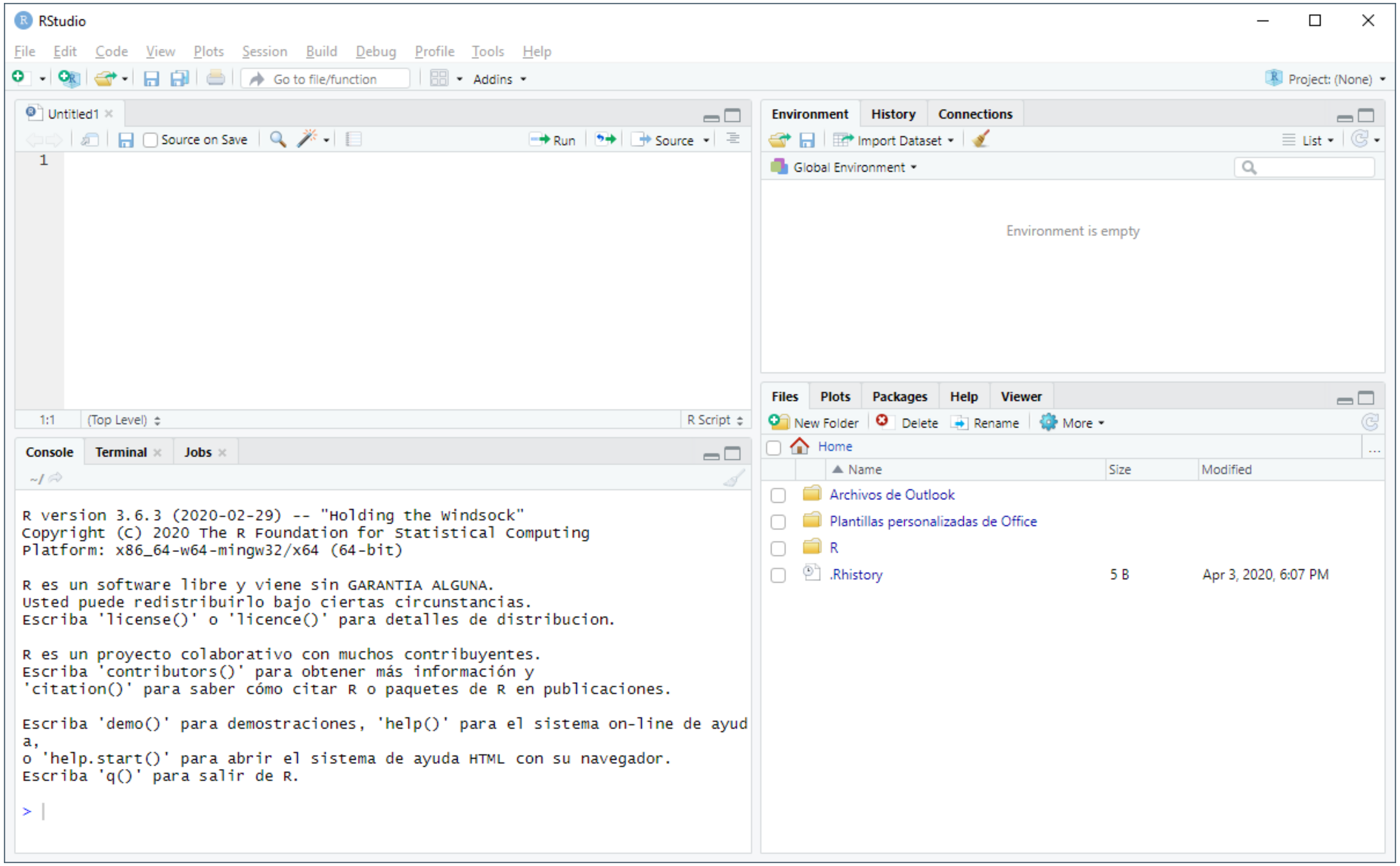
Interface R



```
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> 
```

Exercise 2

- R Studio installs (http://www.rstudio.com/)



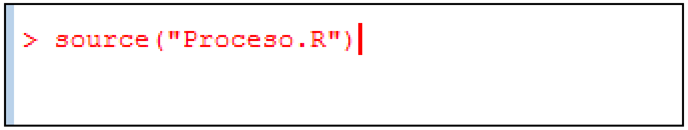
```
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayud
a,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> |
```

Interface R

- Console: Interactive Commands are entered directly
  - Good to look the way they look data
  - Try things
  - Display graphics

Interface R

- Scripts: Files containing reproducible R code in the console
  - Files with extension .R

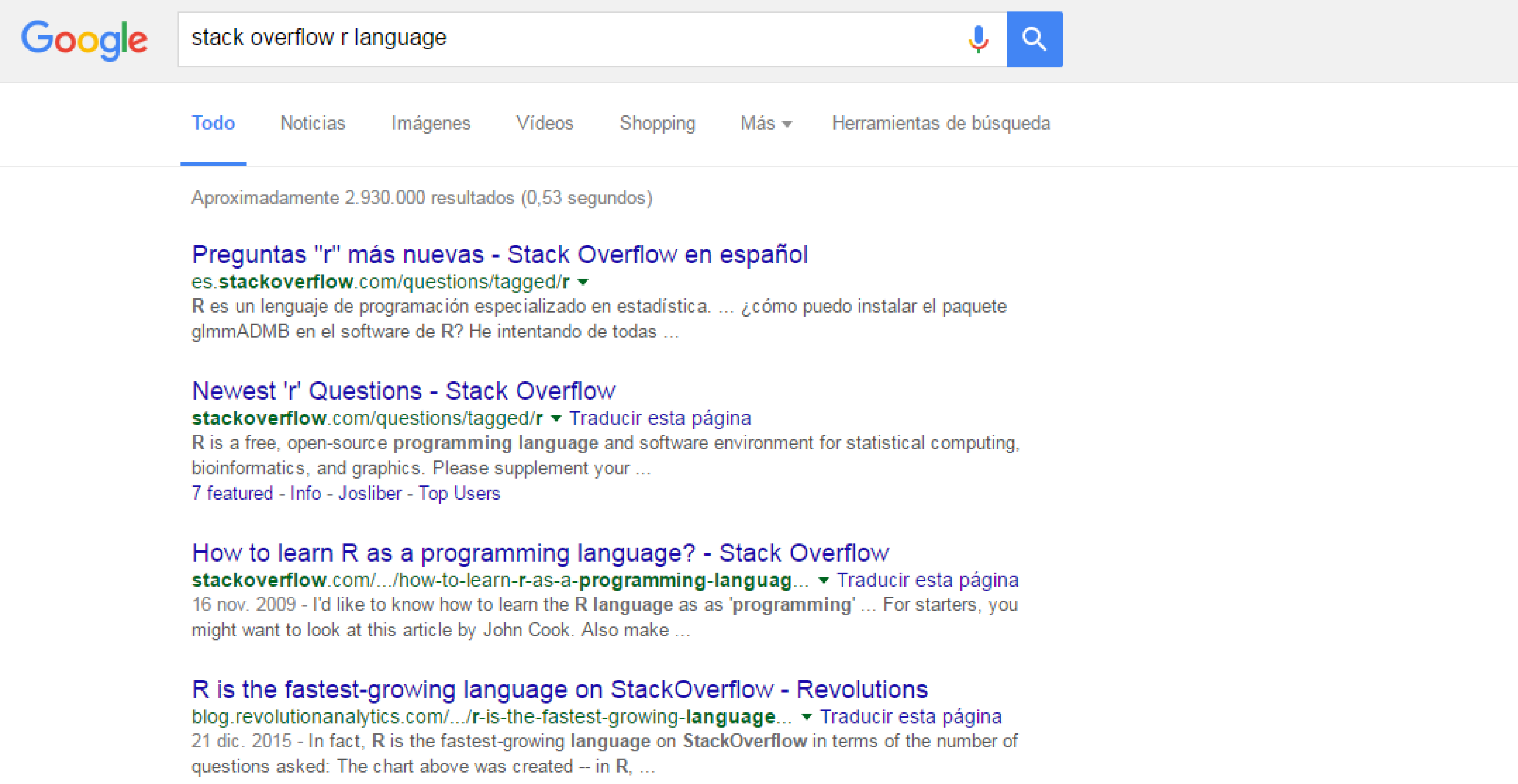


```
> source("Proceso.R")
```

Getting Help

- **help.start**()
  - General Help
- **help**("mean")
  - specific help function
  - ? Mean
- **help.search**(“mean")
  - Find a function on any page Help
  - ?? mean
- **example**(mean)
  - Mostar an example of use of a function

Help - StackOverflow



Aproximadamente 2.930.000 resultados (0,53 segundos)

Preguntas "r" más nuevas - Stack Overflow en español
es.stackoverflow.com/questions/tagged/r
R es un lenguaje de programación especializado en estadística. ... ¿cómo puedo instalar el paquete glmmADMB en el software de R? He intentando de todas ...

Newest 'r' Questions - Stack Overflow
stackoverflow.com/questions/tagged/r Traducir esta página
R is a free, open-source programming language and software environment for statistical computing, bioinformatics, and graphics. Please supplement your ...
7 featured - Info - Josliber - Top Users

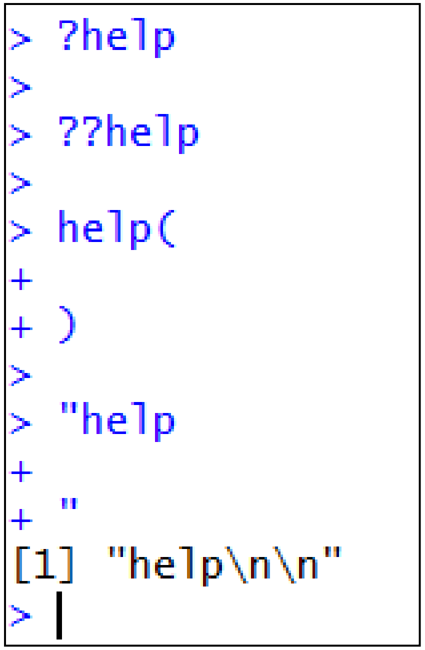
How to learn R as a programming language? - Stack Overflow
stackoverflow.com/.../how-to-learn-r-as-a-programming-languag... Traducir esta página
16 nov. 2009 - I'd like to know how to learn the R language as as 'programming' ... For starters, you might want to look at this article by John Cook. Also make ...

R is the fastest-growing language on StackOverflow - Revolutions
blog.revolutionanalytics.com/.../r-is-the-fastest-growing-language... Traducir esta página
21 dic. 2015 - In fact, R is the fastest-growing language on StackOverflow in terms of the number of questions asked: The chart above was created -- in R, ...

Exercise 3

- Ask for help about ‘help' function
- Find the help pages where reference is made to this function
- Try write in the console help(
- Try write in the console "help

Exercise 3 (Solution)



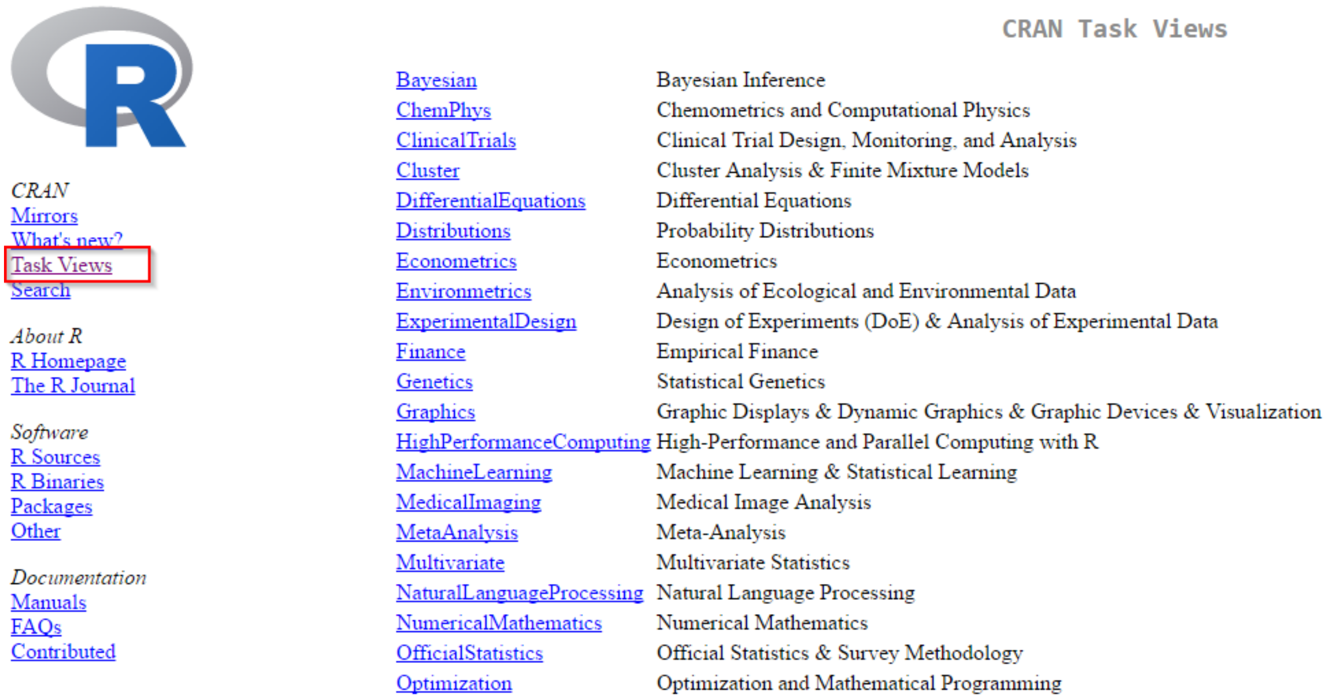
```
> ?help
> 
> ??help
> 
> help(
+ 
+ )
> 
> "help
+ 
+ "
[1] "help\n\n"
> 
```

Packages

- Packages are collections of functions and data in R
- R comes with a standard set of packages
- Many more are available for download and installation
- Once installed, they must be charged before we can use them

Locate a Package

- It is easy to search packages through the tasks view CRAN



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

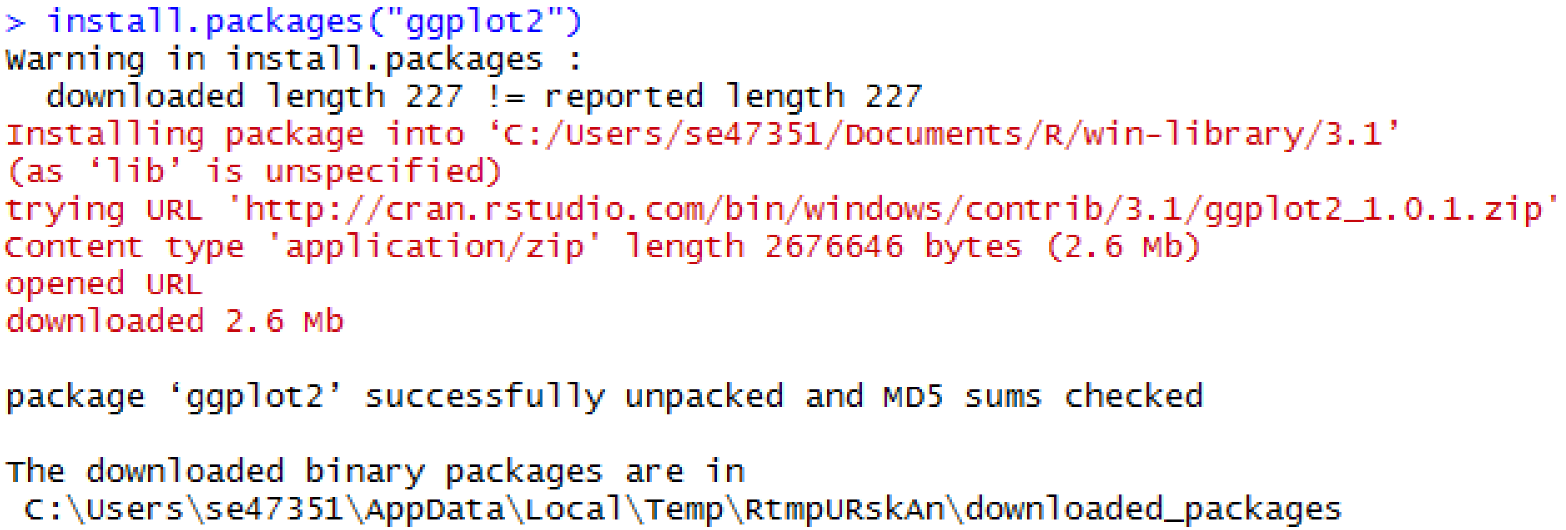
CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming

https://cran.r-project.org/

Installing a new package

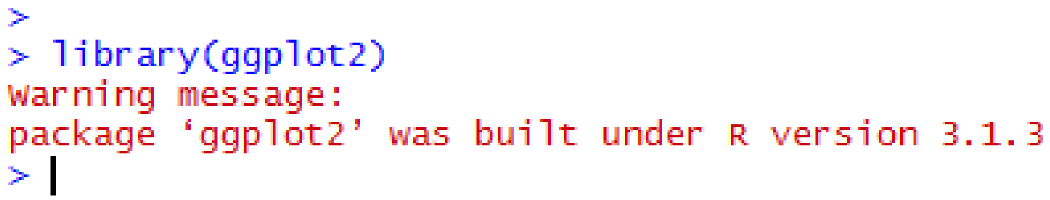
- The function **install.packages()** It is used to install new packages



```
> install.packages("ggplot2")
Warning in install.packages :
  downloaded length 227 != reported length 227
Installing package into ‘C:/Users/se47351/Documents/R/win-library/3.1’
(as ‘lib’ is unspecified)
trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/ggplot2_1.0.1.zip'
Content type 'application/zip' length 2676646 bytes (2.6 Mb)
opened URL
downloaded 2.6 Mb

package ‘ggplot2’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
 C:\Users\se47351\AppData\Local\Temp\RtmpURskAn\downloaded_packages
```

Using a package

- You need to use the function **library()** before using a package that is not within the package base R
- In this way all functions and data contained in the library are loaded into memory



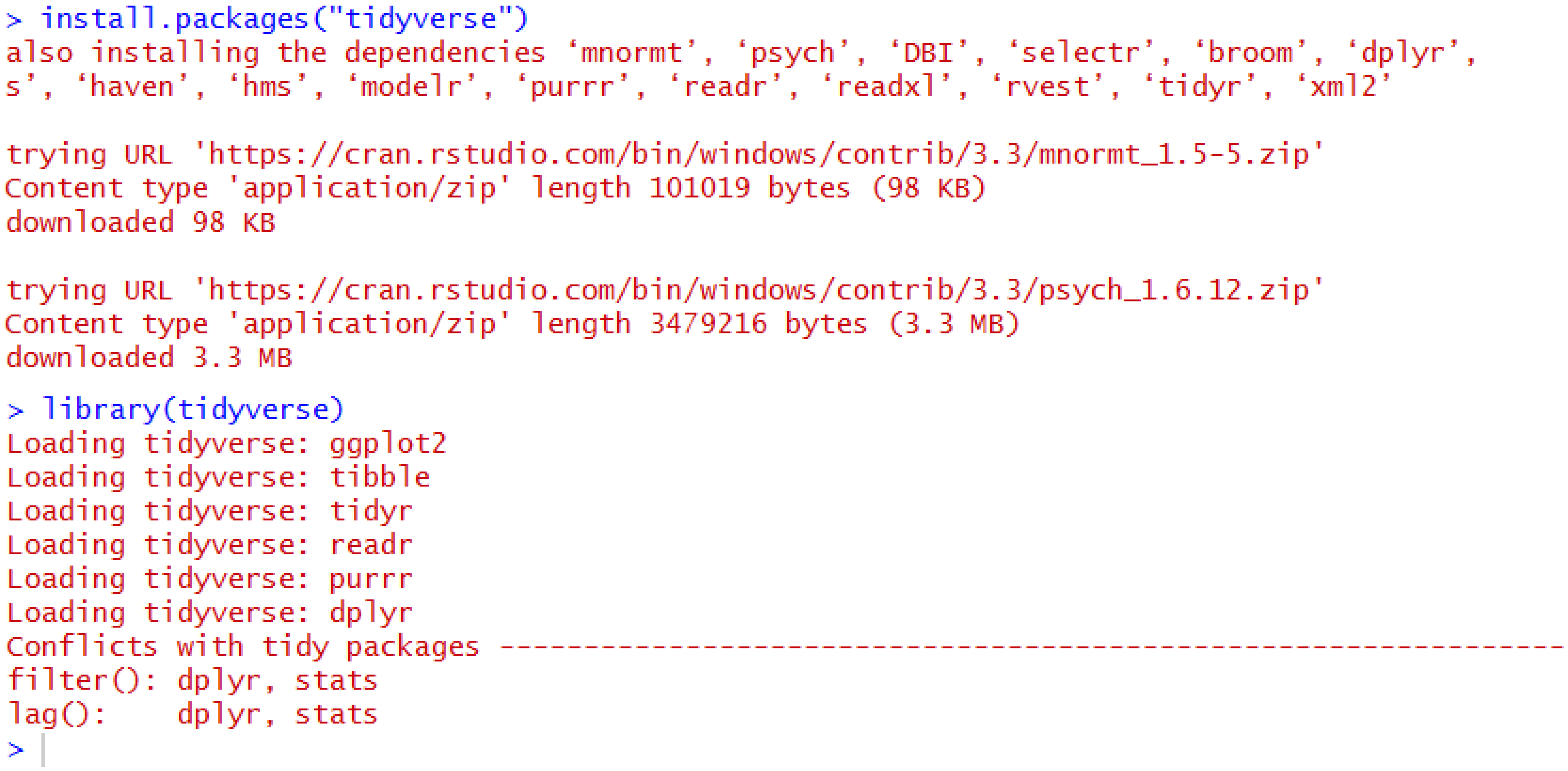
```
>
> library(ggplot2)
Warning message:
package ‘ggplot2’ was built under R version 3.1.3
> |
```

Exercise 4

- Install a package called “tidyverse”
- Load into memory functions contained in this library

https://www.tidyverse.org/

Exercise 4 (Solution)



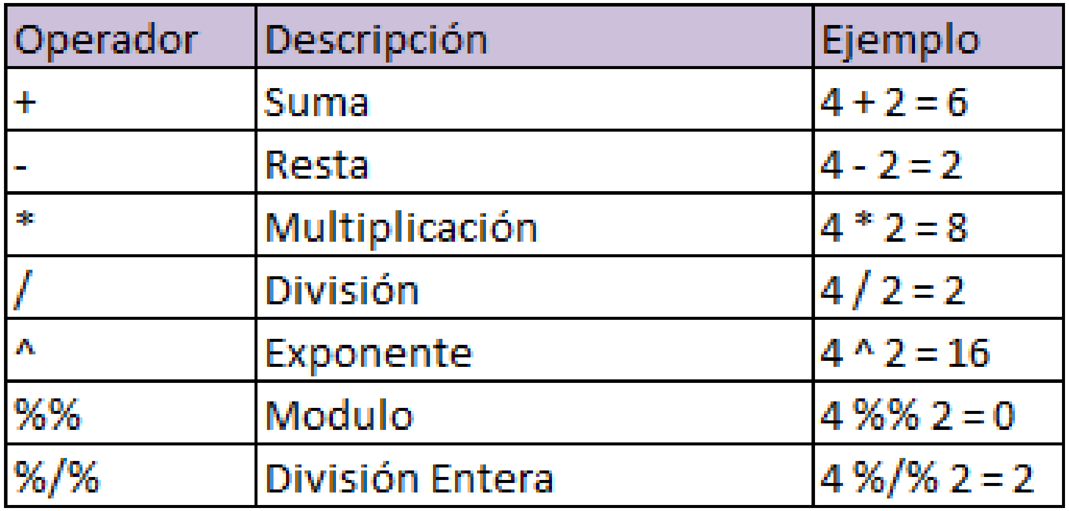
```
> install.packages("tidyverse")
also installing the dependencies ‘mnormt’, ‘psych’, ‘DBI’, ‘selectr’, ‘broom’, ‘dplyr’,
s’, ‘haven’, ‘hms’, ‘modelr’, ‘purrr’, ‘readr’, ‘readxl’, ‘rvest’, ‘tidyr’, ‘xml2’

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/mnormt_1.5-5.zip'
Content type 'application/zip' length 101019 bytes (98 KB)
downloaded 98 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/psych_1.6.12.zip'
Content type 'application/zip' length 3479216 bytes (3.3 MB)
downloaded 3.3 MB

> library(tidyverse)
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
Conflicts with tidy packages ---------------------------------------------------------------
filter(): dplyr, stats
lag():    dplyr, stats
> |
```

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

R as a calculator

- R console functions as a calculator
- R has the following arithmetic operators:

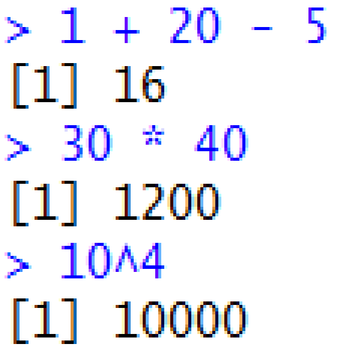


Operador	Descripción	Ejemplo
+	Suma	4 + 2 = 6
-	Resta	4 - 2 = 2
*	Multiplicación	4 * 2 = 8
/	División	4 / 2 = 2
^	Exponente	4 ^ 2 = 16
%%	Modulo	4 %% 2 = 0
%/%	División Entera	4 %/% 2 = 2

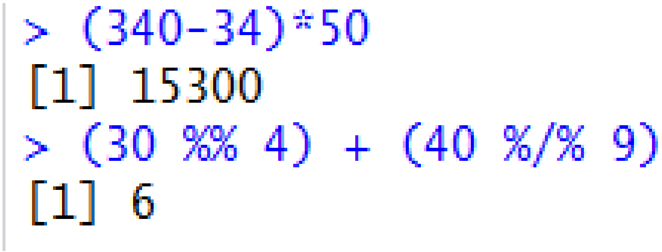
Exercise 5

- Prints the result of the following operations:
  - 1 plus 20 minus 5 (16)
  - 30 multiplied by 40 (1200)
  - 10 to the 4th power (10,000)
  - Multiply the result of substracting 34 from 340 by 50 (15,300)
  - Add the remainder of dividing 30 by 4 to the result of dividing 40 by 9 (using the integer division) (6)

Exercise 5 (Solution)



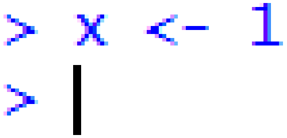
```
> 1 + 20 - 5
[1] 16
> 30 * 40
[1] 1200
> 10^4
[1] 10000
```



```
> (340-34)*50
[1] 15300
> (30 %% 4) + (40 %/% 9)
[1] 6
```

Variables

- A variable allows us to use a name to store a value
- The assignment operator **<-** assigns a value to a variable
- If a variable name is capitalized is not the same in lower case (case R is sensitive)



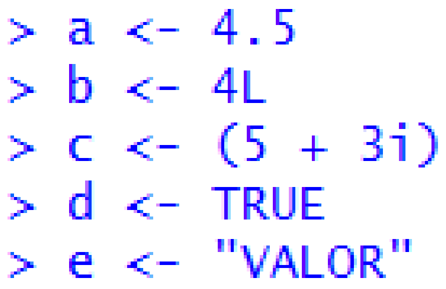
```
> x <- 1
> 
```

Type of data

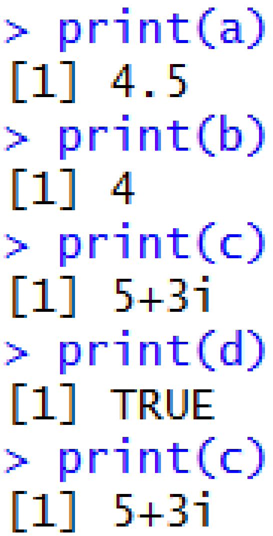
- R has 5 types of data:
  - decimal values as 4.5
  - integers such as 4
    - You can specify that a value is an integer with the suffix L
  - complex numbers as (5 + 3i)
  - boolean values (TRUE or FALSE)
    - It may be abbreviated to T or F
  - Text values
    - Quotation marks are used to indicate that a value is text

Finding out the type of a variable

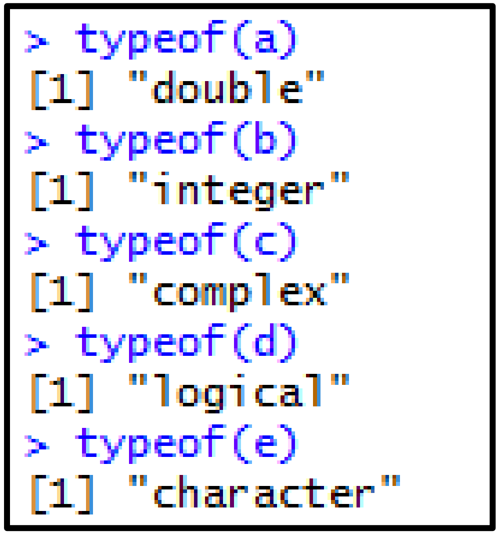
- The function **typeof()** gives information on the particular type



```
> a <- 4.5
> b <- 4L
> c <- (5 + 3i)
> d <- TRUE
> e <- "VALOR"
```



```
> print(a)
[1] 4.5
> print(b)
[1] 4
> print(c)
[1] 5+3i
> print(d)
[1] TRUE
> print(c)
[1] 5+3i
```

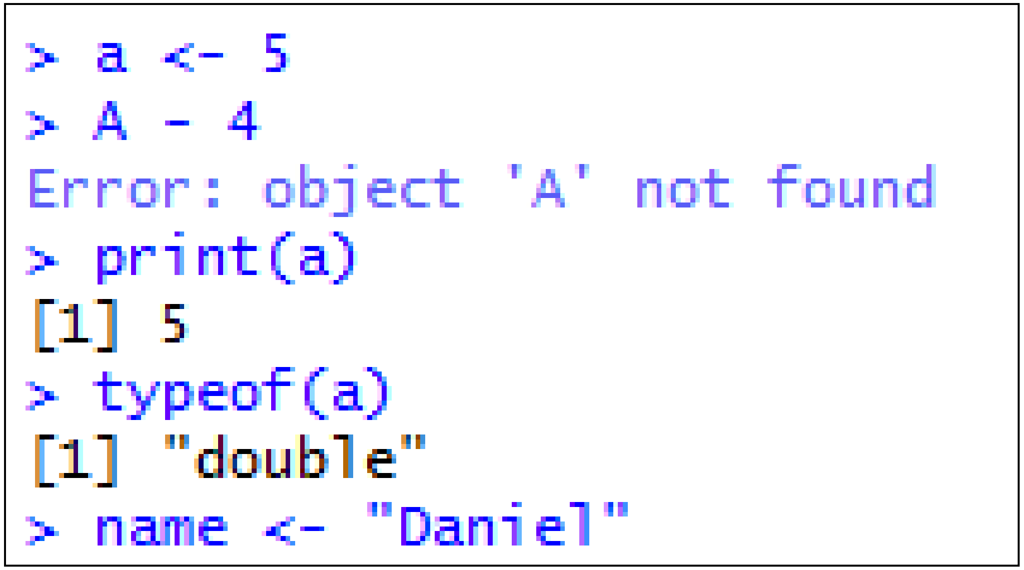


```
> typeof(a)
[1] "double"
> typeof(b)
[1] "integer"
> typeof(c)
[1] "complex"
> typeof(d)
[1] "logical"
> typeof(e)
[1] "character"
```

Exercise 6

- Assigns the value 5 to the variable “**a**”
- Subtract 4 to the variable “**A**”
- Prints the value of the variable “**a**”
- Prints its type
- Assign your name to the variable “**name**”

Exercise 6 (Solution)



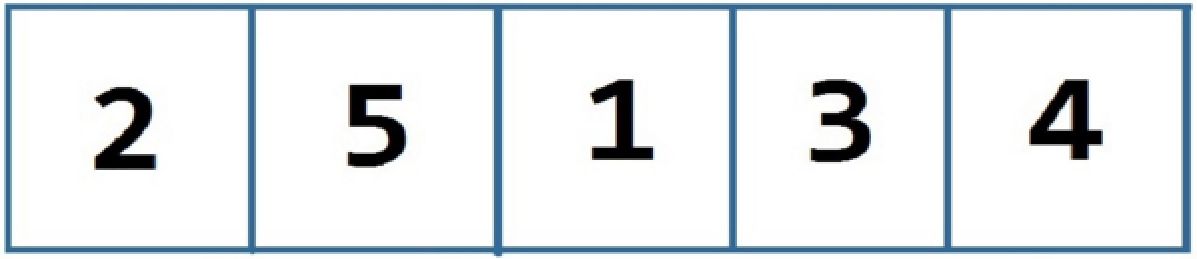
```
> a <- 5
> A - 4
Error: object 'A' not found
> print(a)
[1] 5
> typeof(a)
[1] "double"
> name <- "Daniel"
```

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Vectors

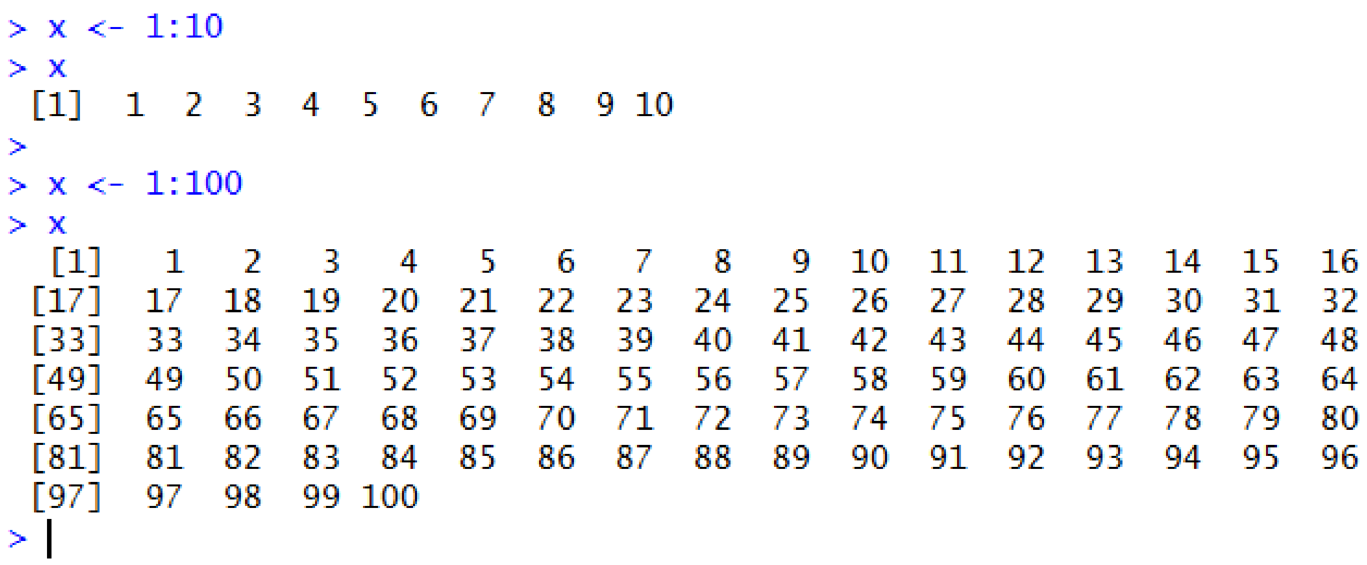
- The most basic structure data with which operates R is a **vector**
- A vector is a collection of data of the same type



2	5	1	3	4

Create a vector

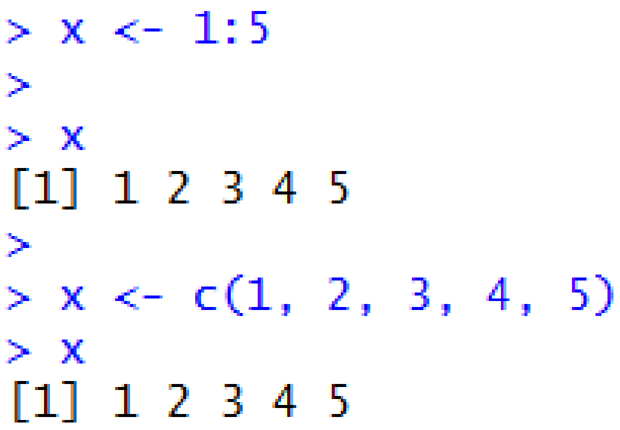
- The operator **:** is used to create vectors number sequences



```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> x <- 1:100
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 [17]  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
 [33]  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
 [49]  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
 [65]  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
 [81]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
 [97]  97  98  99 100
> |
```

Create a vector

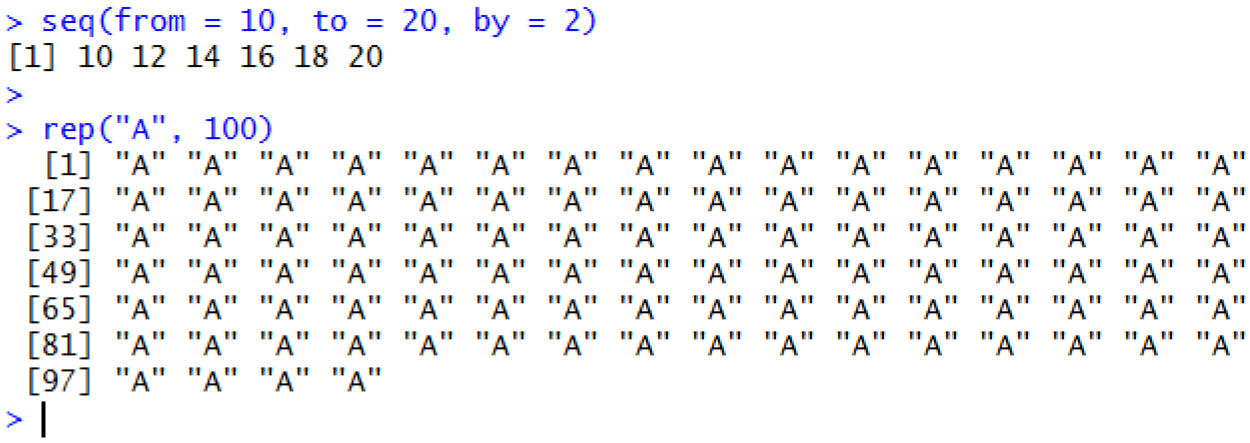
- The function **c()** It is used to create a vector with a list of elements of the same type



```
> x <- 1:5
>
> x
[1] 1 2 3 4 5
>
> x <- c(1, 2, 3, 4, 5)
> x
[1] 1 2 3 4 5
```

Create a vector

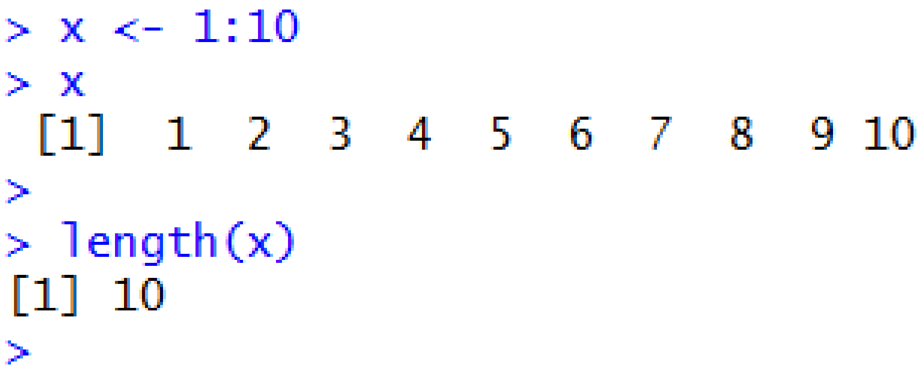
- The function **seq()** is used to create sequences of numbers
- But if you need a table with a repetition of values use **rep()**



```
> seq(from = 10, to = 20, by = 2)
[1] 10 12 14 16 18 20
>
> rep("A", 100)
  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [17] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [33] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [49] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [65] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [81] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [97] "A" "A" "A" "A"
> |
```

Length of a vector

- The function **length()** It allows for the elements of a vector

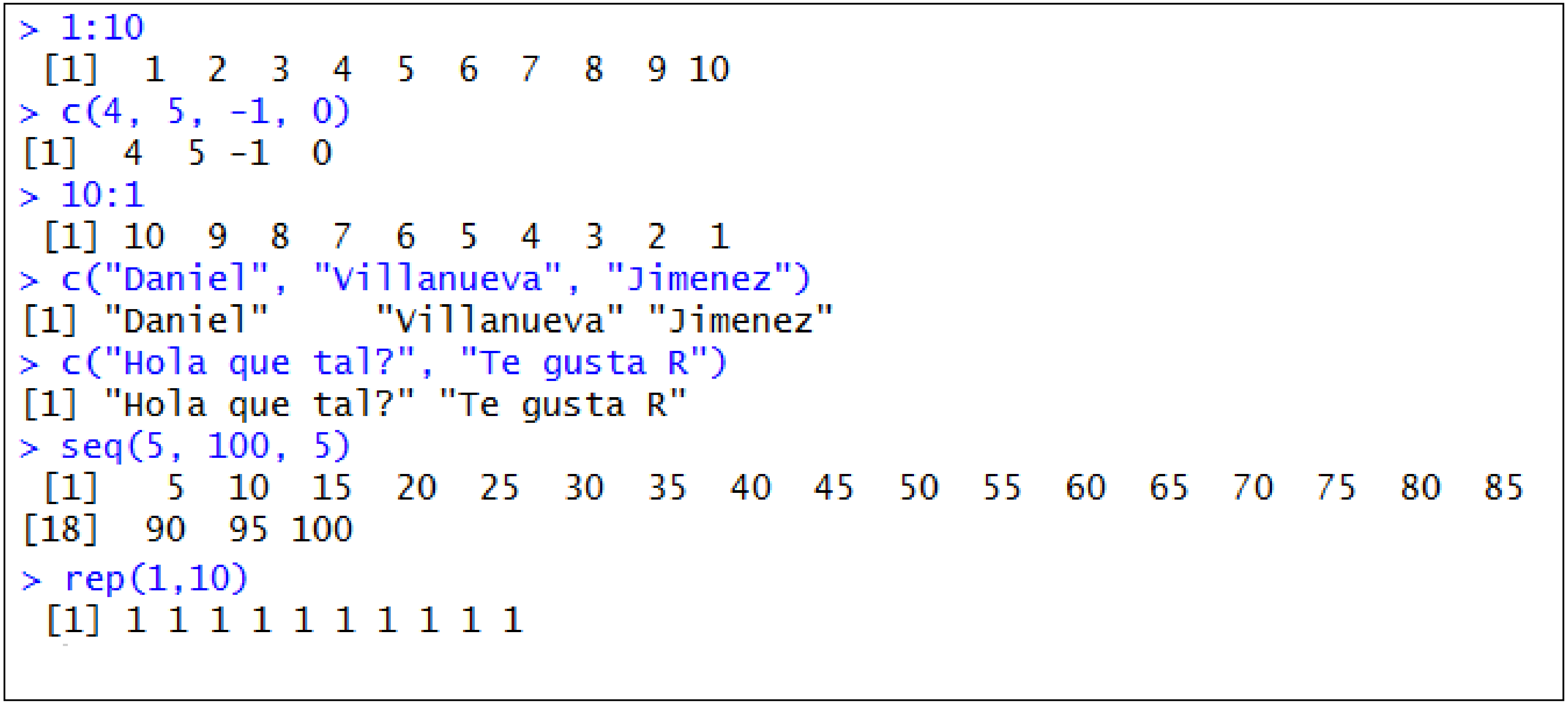


```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> length(x)
[1] 10
>
```

Exercise 7

- Create a vector of numbers from 1 to 10
- Create a vector with the numbers 4, 5, 1, -1 and 0
- Create a vector of numbers from 10 to 1
- Create a vector of words with your full name
- Create a vector with 2 sentences
- Create a vector with numbers from 5 to 100 containing only multiples of 5
- Create a table of 10 elements all filled with the number 1 and print its size

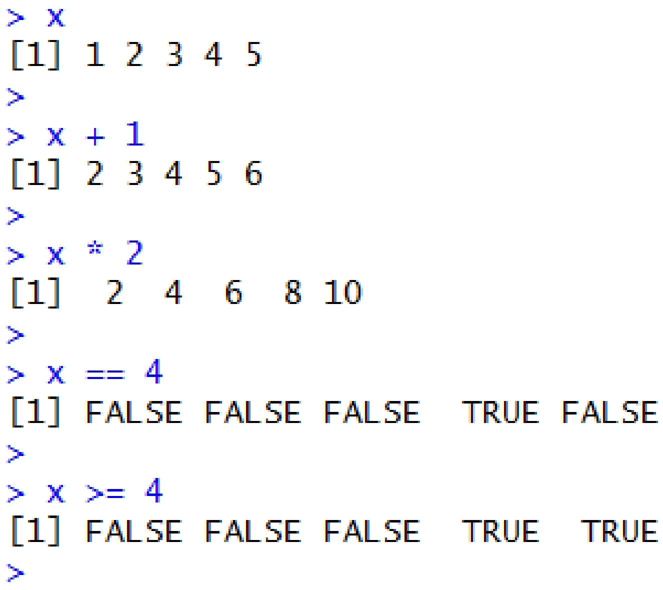
Exercise 7 (Solution)



```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> c(4, 5, -1, 0)
[1]  4  5 -1  0
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
> c("Daniel", "Villanueva", "Jimenez")
[1] "Daniel"     "Villanueva" "Jimenez"
> c("Hola que tal?", "Te gusta R")
[1] "Hola que tal?" "Te gusta R"
> seq(5, 100, 5)
 [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85
[18]  90  95 100
> rep(1,10)
 [1] 1 1 1 1 1 1 1 1 1 1
```

Operate with a vector

- R operations are vectorized
- If an operation is performed with a vector, the operation is applied to all elements

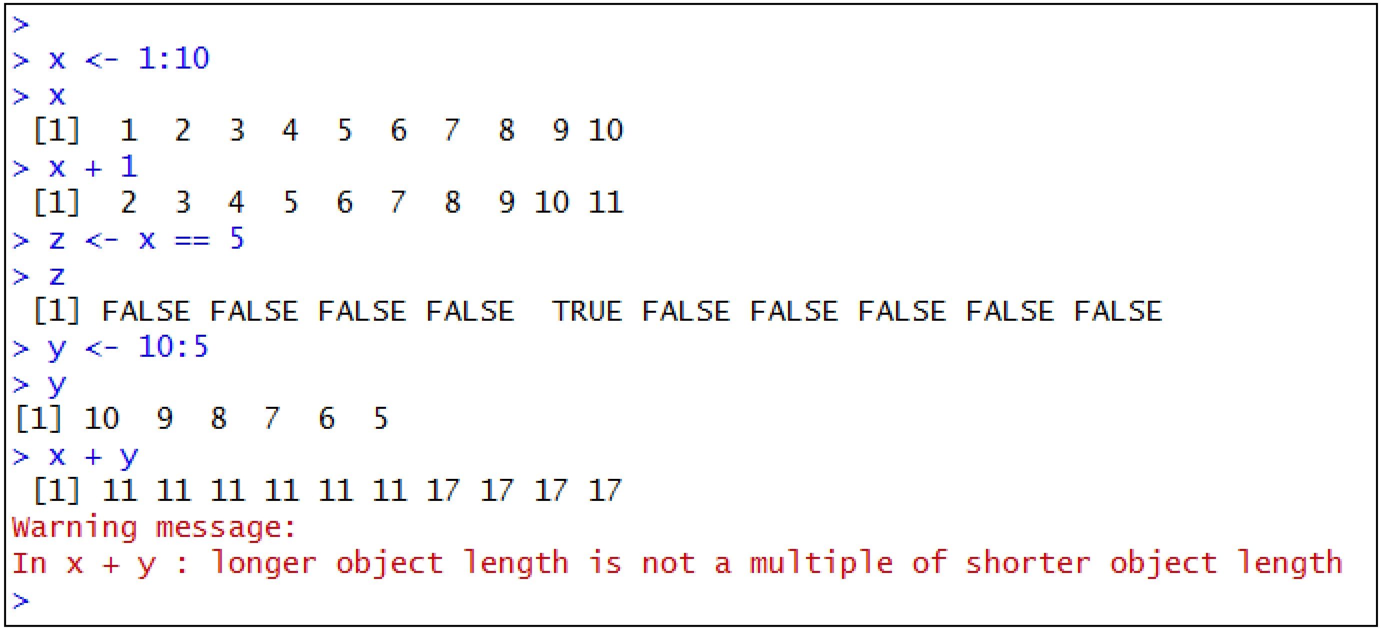


```
> x
[1] 1 2 3 4 5
>
> x + 1
[1] 2 3 4 5 6
>
> x * 2
[1]  2  4  6  8 10
>
> x == 4
[1] FALSE FALSE FALSE  TRUE FALSE
>
> x >= 4
[1] FALSE FALSE FALSE  TRUE  TRUE
>
```

Exercise 8

- Create a table with numbers 1 through 10 and assign it to the variable "x"
- Add 1 to the vector "x"
- Compare the vector "x" with the number 5 and assign the result to vector "z"
- Create a vector with numbers from 10 to 5 and assign it to the variable "y"
- Add the vector "x" and the vector "y" (+)

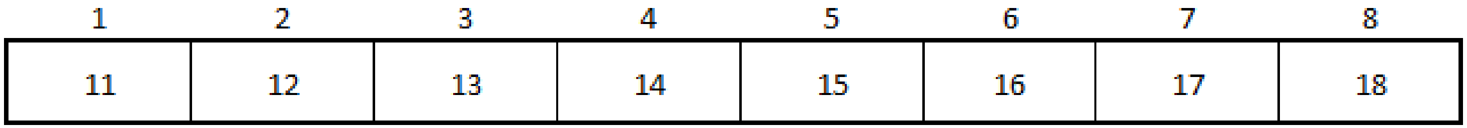
Exercise 8 (Solution)



```
>
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x + 1
 [1]  2  3  4  5  6  7  8  9 10 11
> z <- x == 5
> z
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
> y <- 10:5
> y
[1] 10  9  8  7  6  5
> x + y
 [1] 11 11 11 11 11 11 17 17 17 17
Warning message:
In x + y : longer object length is not a multiple of shorter object length
>
```

Selecting from a vector

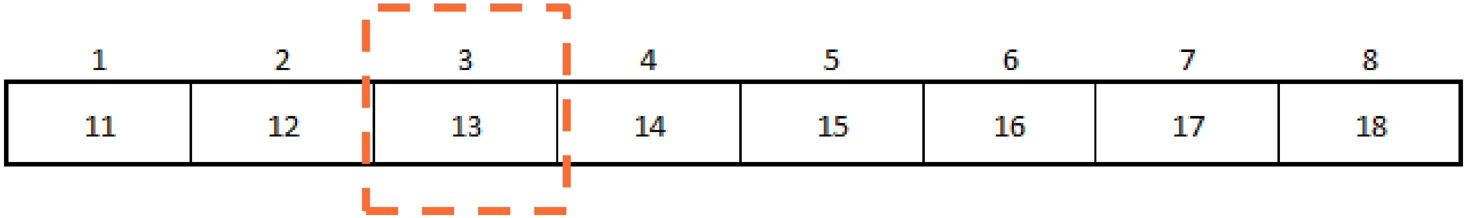
- To select a vector element through its index symbols are used **[** Y **]**



1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18

Selecting from a vector

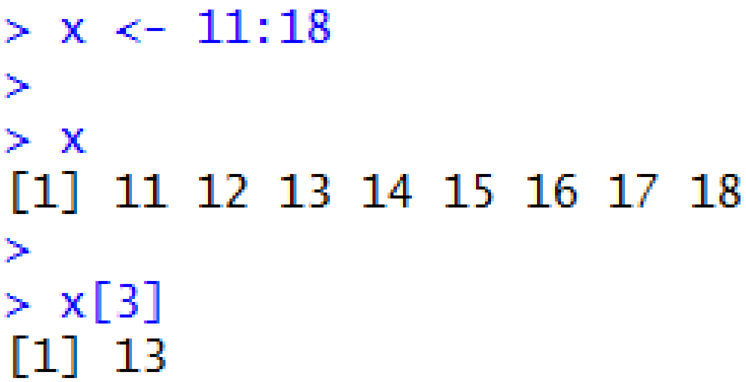
- To select a vector element through its index symbols are used **[** Y **]**



1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18

Selecting from a vector

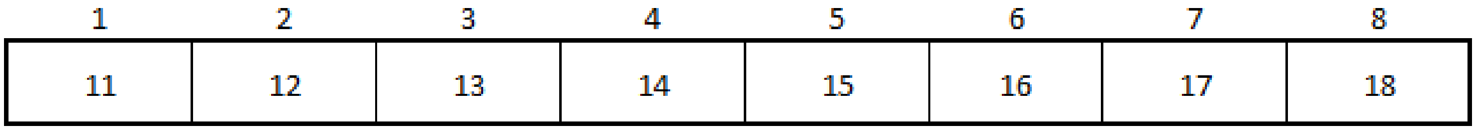
- To select a vector element through its index symbols are used **[Y]**



```
> x <- 11:18
>
> x
[1] 11 12 13 14 15 16 17 18
>
> x[3]
[1] 13
```

Selecting from a vector

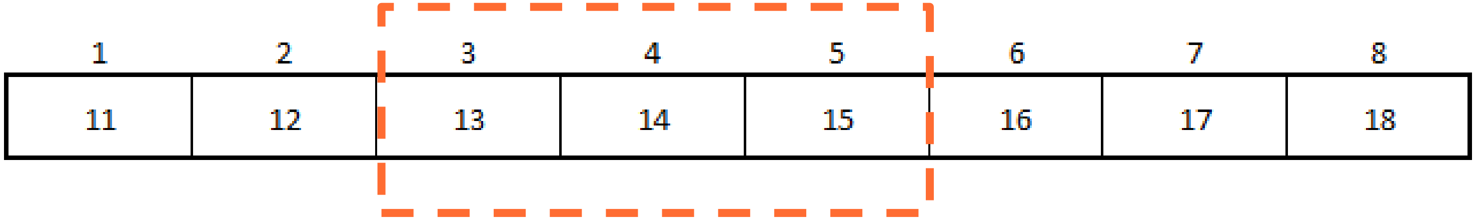
- You can select more than one item ..



1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18

Selecting from a vector

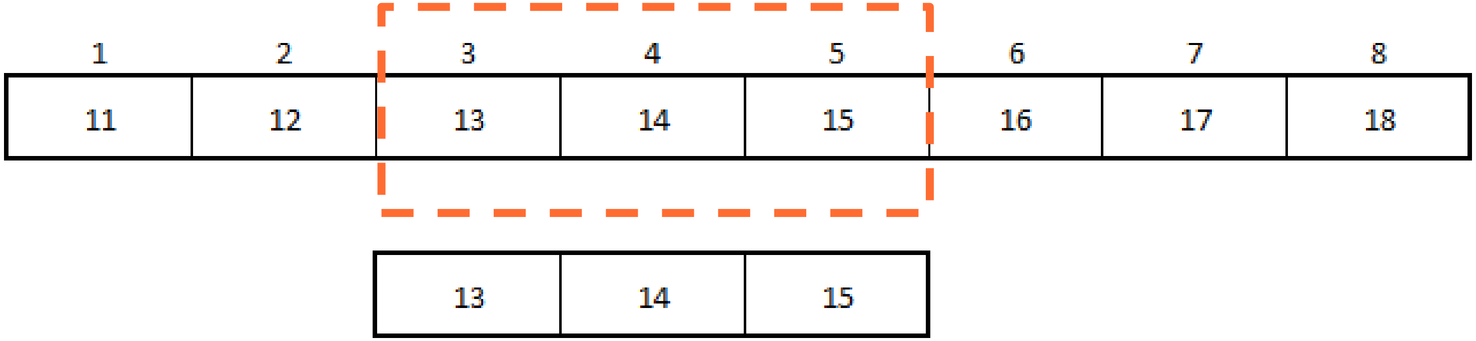
- You can select more than one item …



1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18

Selecting from a vector

- You can select more than one item …

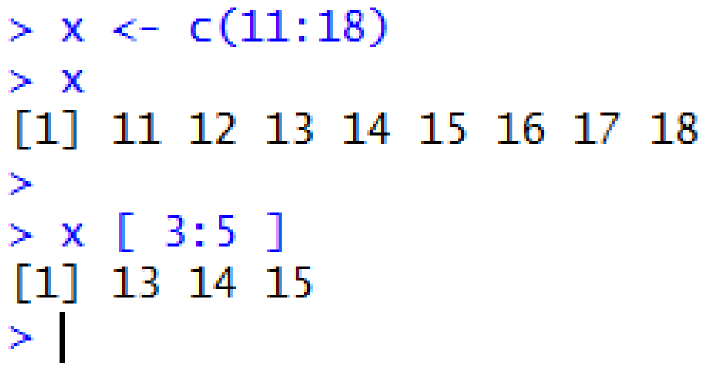


1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18

13	14	15

Selecting from a vector

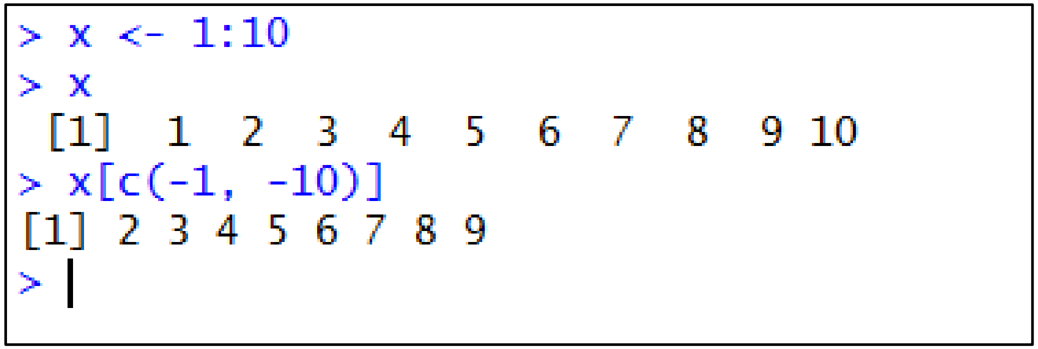
- You can select more than one item …



```
> x <- c(11:18)
> x
[1] 11 12 13 14 15 16 17 18
>
> x [ 3:5 ]
[1] 13 14 15
> |
```

Selecting from a vector

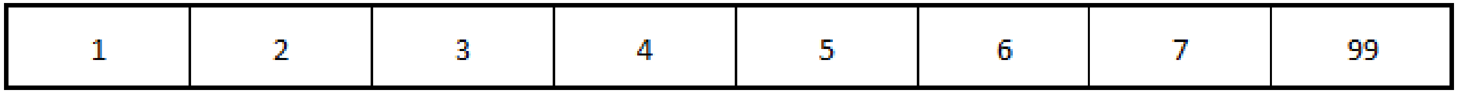
- If we put a negative number instead of a positive one, we are asking R to exclude a position (but include all the others)



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

Selecting from a vector

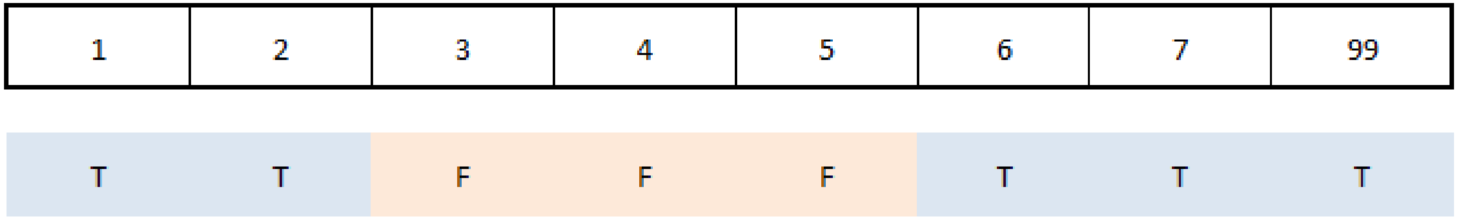
- R can be selected elements of a vector through another vector of logic values



1	2	3	4	5	6	7	99

Selecting from a vector

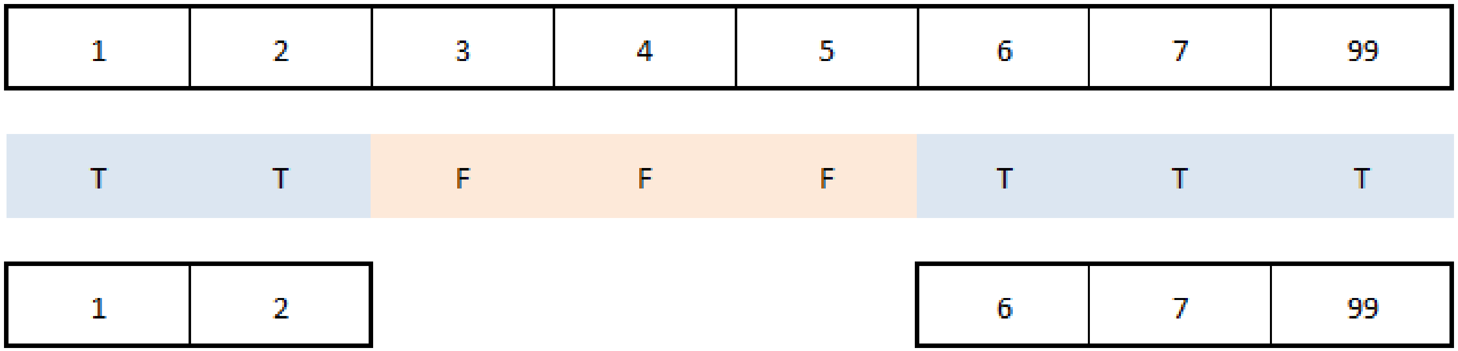
- In R can be selected elements of a vector through another vector of logic values



1	2	3	4	5	6	7	99
T	T	F	F	F	T	T	T

Selecting from a vector

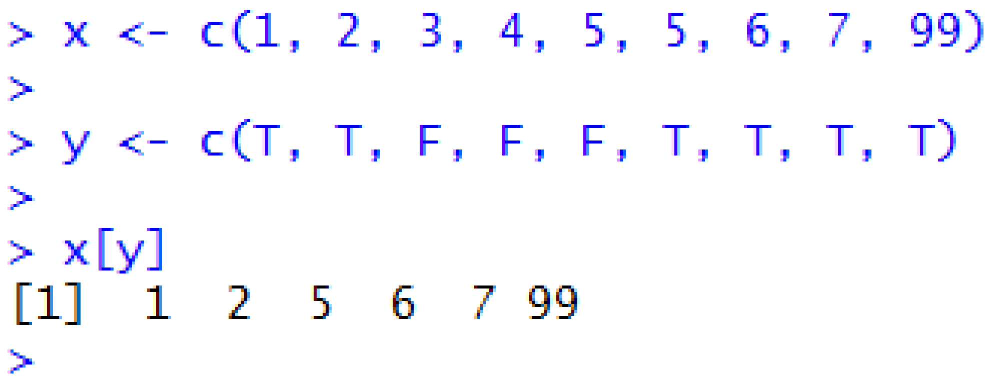
- In R can be selected elements of a vector through another vector of logic values



1	2	3	4	5	6	7	99
T	T	F	F	F	T	T	T
1	2				6	7	99

Selecting from a vector

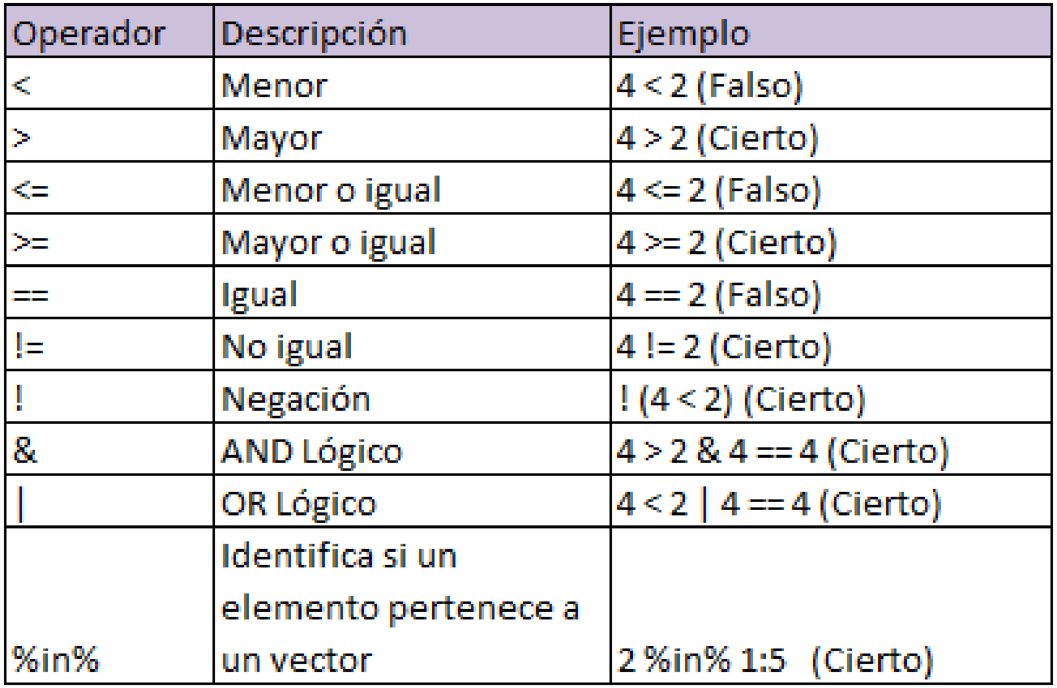
- In R can be selected elements of a vector through another vector of logic values



```
> x <- c(1, 2, 3, 4, 5, 5, 6, 7, 99)
>
> y <- c(T, T, F, F, F, T, T, T, T)
>
> x[y]
[1]  1  2  5  6  7 99
>
```

Logical operations

- R has the following logical operators:

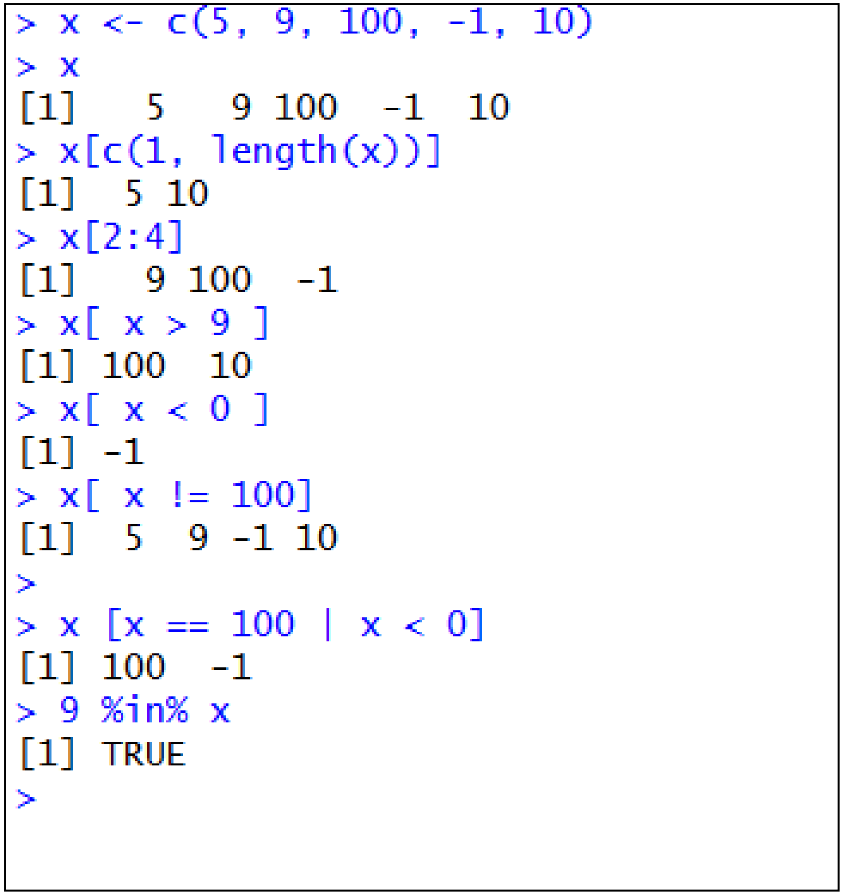


Operador	Descripción	Ejemplo
<	Menor	4 < 2 (Falso)
>	Mayor	4 > 2 (Cierto)
<=	Menor o igual	4 <= 2 (Falso)
>=	Mayor o igual	4 >= 2 (Cierto)
==	Igual	4 == 2 (Falso)
!=	No igual	4 != 2 (Cierto)
!	Negación	! (4 < 2) (Cierto)
&	AND Lógico	4 > 2 & 4 == 4 (Cierto)
|	OR Lógico	4 < 2 | 4 == 4 (Cierto)
%in%	Identifica si un elemento pertenece a un vector	2 %in% 1:5 (Cierto)

Exercise 9

- Create a vector called "x" with the numbers: 5, 9, 100, -1 and 10
- Print the numbers found in the first and last position
- Print 3 central positions "x"
- Print the numbers are greater than 9
- Print numbers less than 0
- Print numbers other than 100
- Print numbers equal to 100 or less than 0
- Check whether the number 9 is on the table "x"

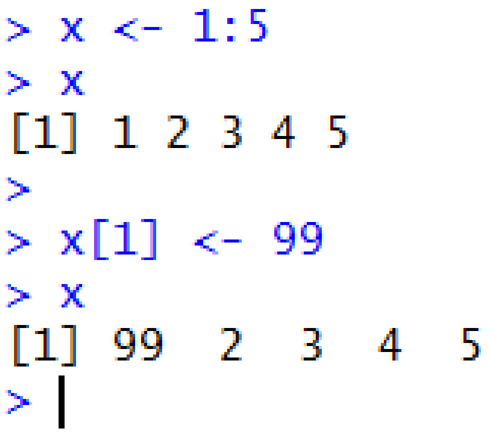
Exercise 9 (Solution)



```
> x <- c(5, 9, 100, -1, 10)
> x
[1]   5   9 100  -1  10
> x[c(1, length(x))]
[1]  5 10
> x[2:4]
[1]   9 100  -1
> x[ x > 9 ]
[1] 100  10
> x[ x < 0 ]
[1] -1
> x[ x != 100]
[1]  5  9 -1 10
>
> x [x == 100 | x < 0]
[1] 100  -1
> 9 %in% x
[1] TRUE
>
```

Replacing elements in a vector

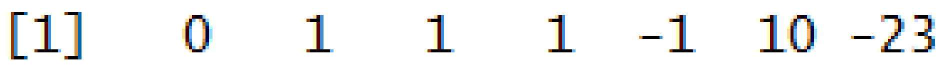
- To replace a particular position of a vector assignment operator is used (**<-**) Along with the selection **[]**



```
> x <- 1:5
> x
[1] 1 2 3 4 5
>
> x[1] <- 99
> x
[1] 99  2  3  4  5
> 
```

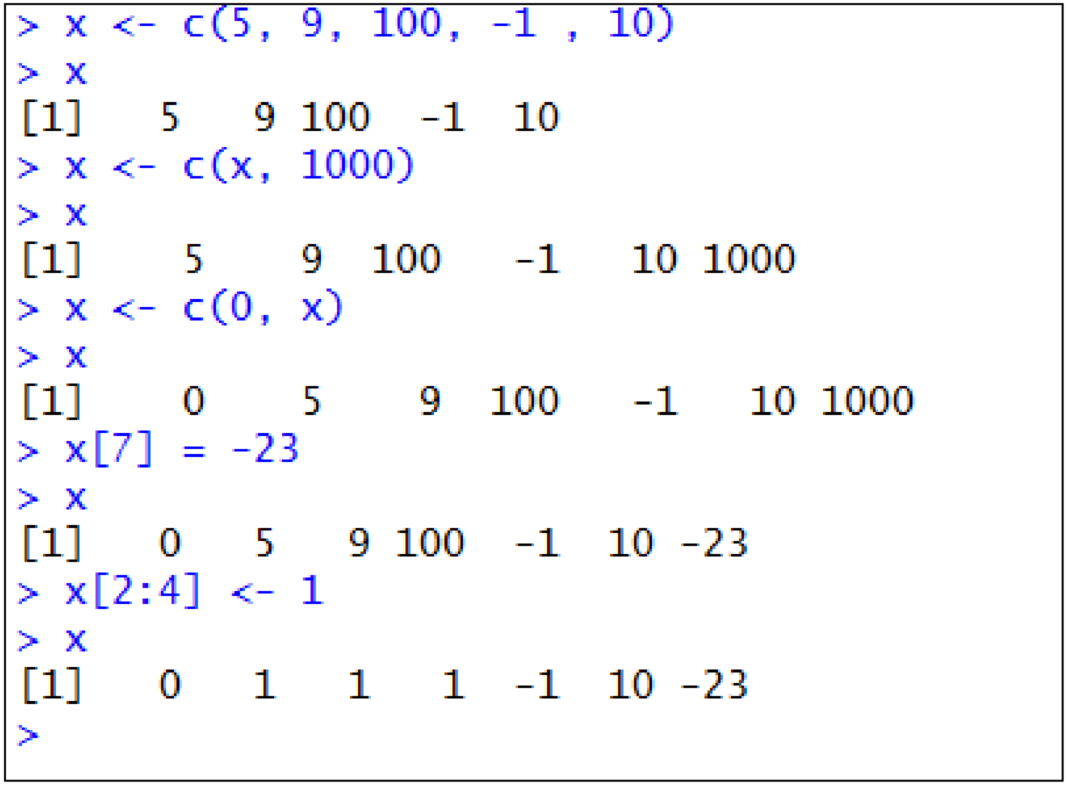
Exercise 10

- Create a vector of numbers 5, 9, 100, -1 and 10 and assign it to the variable "x"
- Add at the end of this vector a new element with the number 1000
- Add an element with the number 0 at the beginning
- Modify position 7 so that its value is -23
- Modify 2, 3 and 4 positions of the vector, so all these positions have 1



```
[1]    0    1    1    1   -1   10  -23
```

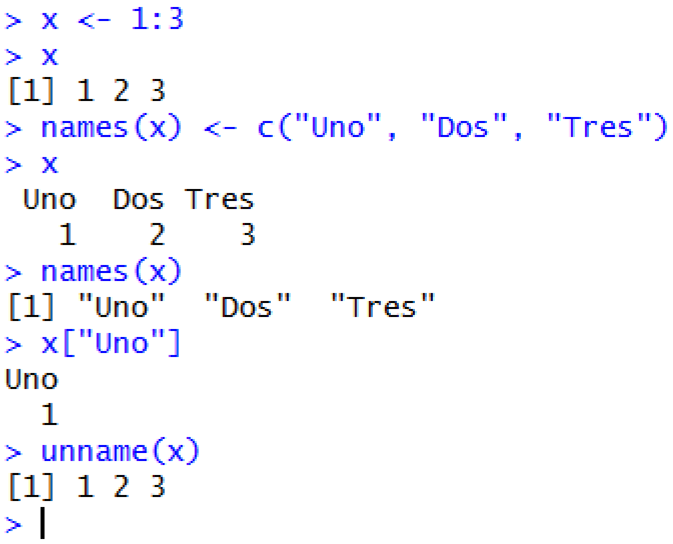
Exercise 10 (Solution)



```
> x <- c(5, 9, 100, -1 , 10)
> x
[1]   5   9 100  -1  10
> x <- c(x, 1000)
> x
[1]    5    9  100   -1   10 1000
> x <- c(0, x)
> x
[1]    0    5    9  100   -1   10 1000
> x[7] = -23
> x
[1]   0   5   9 100  -1  10 -23
> x[2:4] <- 1
> x
[1]   0   1   1   1  -1  10 -23
>
```

Tables with names

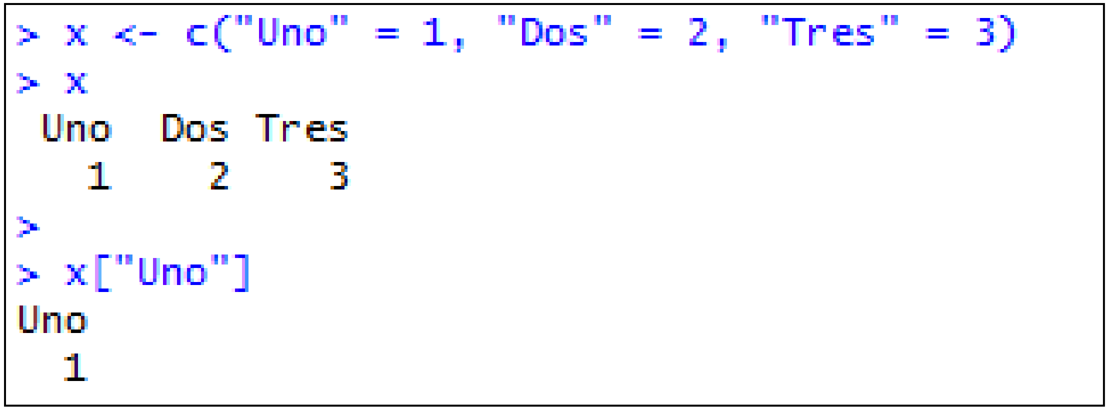
- R may assign a name to each vector element and select items by name
  - Similar to a dictionary



```
> x <- 1:3
> x
[1] 1 2 3
> names(x) <- c("Uno", "Dos", "Tres")
> x
 Uno  Dos Tres
   1    2    3
> names(x)
[1] "Uno"  "Dos"  "Tres"
> x["Uno"]
Uno
  1
> unname(x)
[1] 1 2 3
> 
```

Vector with names

- Once a vector has elements name can be selected through this name

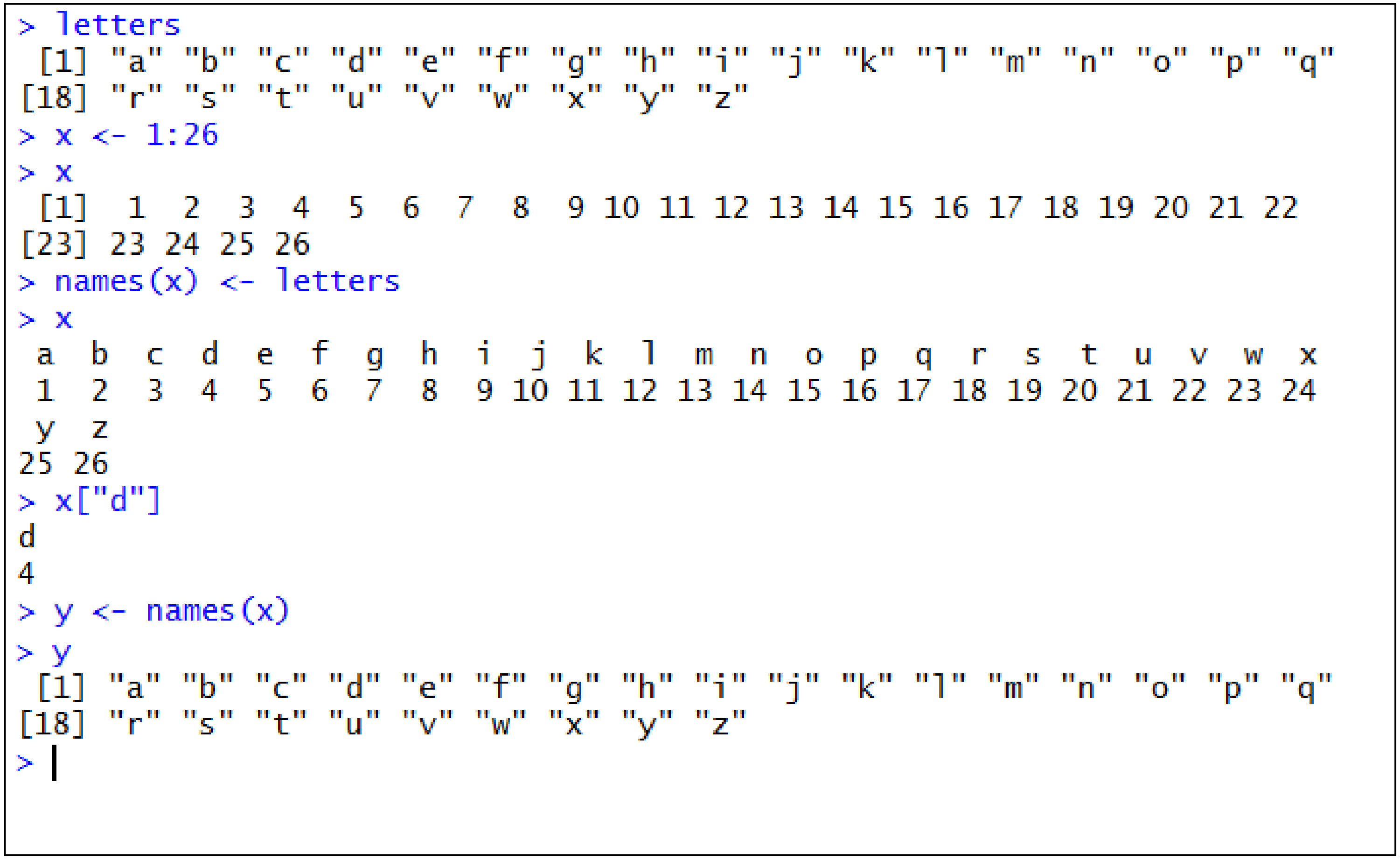


```
> x <- c("Uno" = 1, "Dos" = 2, "Tres" = 3)
> x
 Uno  Dos Tres
   1    2    3
>
> x["Uno"]
Uno
  1
```

Exercise 11

- Print the value of the vector called "letters"
- Create a table called "x" with the numbers 1 to 26
- Name the elements of the vector "x" with the letters of the alphabet
- Print the value that is in the “d” position
- Assign to a vector called “y" the names of vector "x"

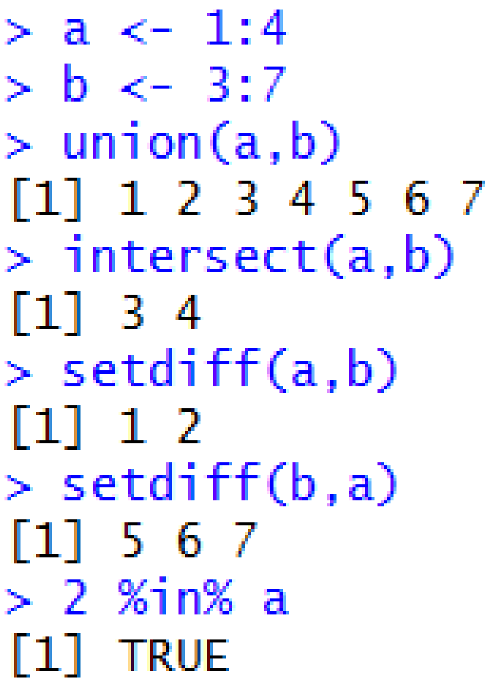
Exercise 11 (Solution)



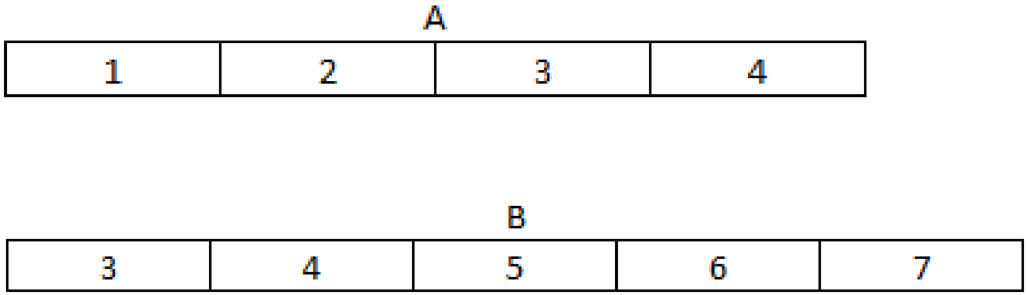
```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
> x <- 1:26
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26
> names(x) <- letters
> x
 a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x 
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 
 y  z 
25 26 
> x["d"]
d 
4 
> y <- names(x)
> y
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
> |
```

Set operations

- R has a series of operations to perform operations on vector sets



```
> a <- 1:4
> b <- 3:7
> union(a,b)
[1] 1 2 3 4 5 6 7
> intersect(a,b)
[1] 3 4
> setdiff(a,b)
[1] 1 2
> setdiff(b,a)
[1] 5 6 7
> 2 %in% a
[1] TRUE
```



A

1	2	3	4

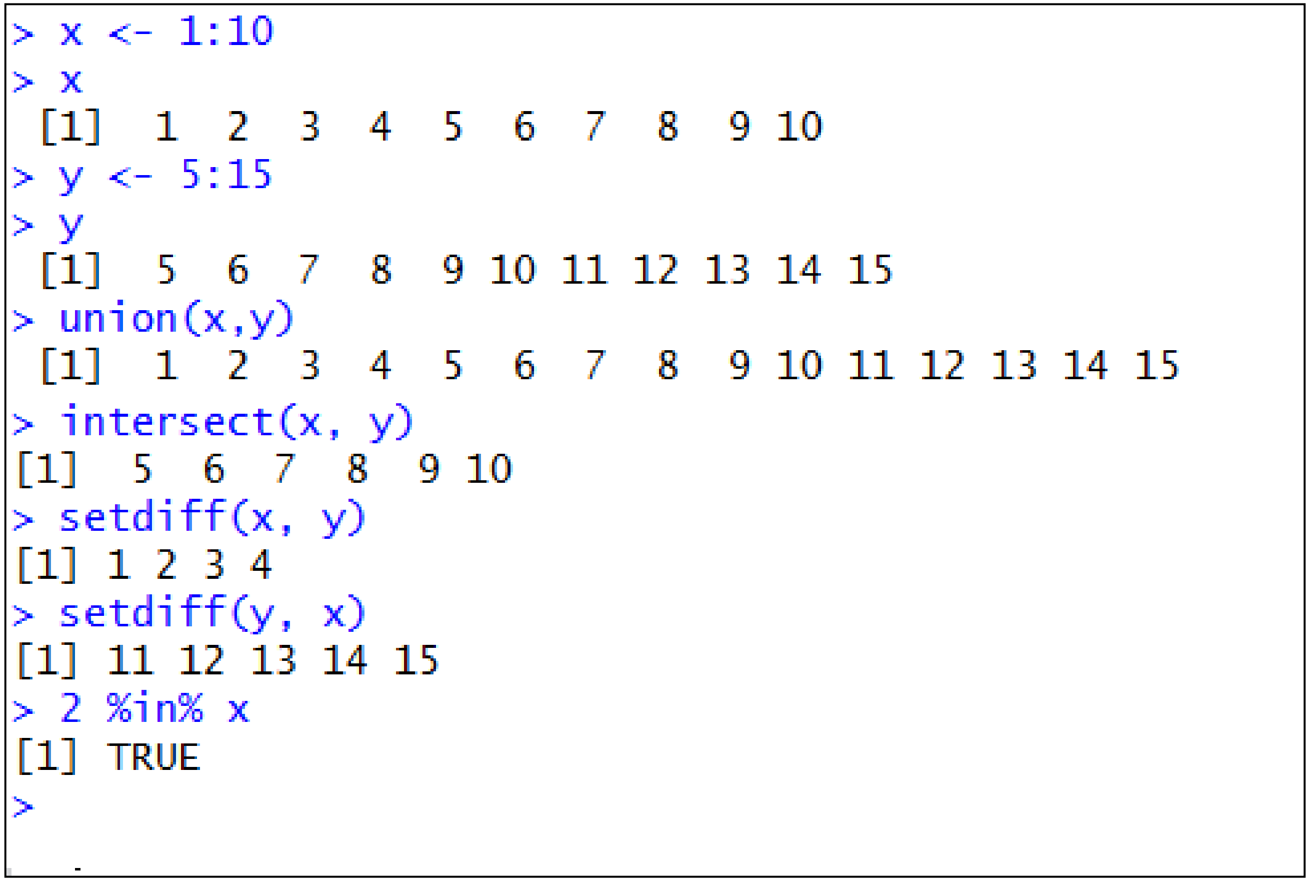
B

3	4	5	6	7

Exercise 12

- Create a vector called "x" with the numbers from 1 to 10
- Create another vector called «and» with the numbers from 5 to 15
- Print the union of "x" and "y"
- Print intersection
- Print the elements that are in «x» but not in «y»
- Print the elements that are in «y» but not in «x»
- Check if the number 2 is in the vector "x"

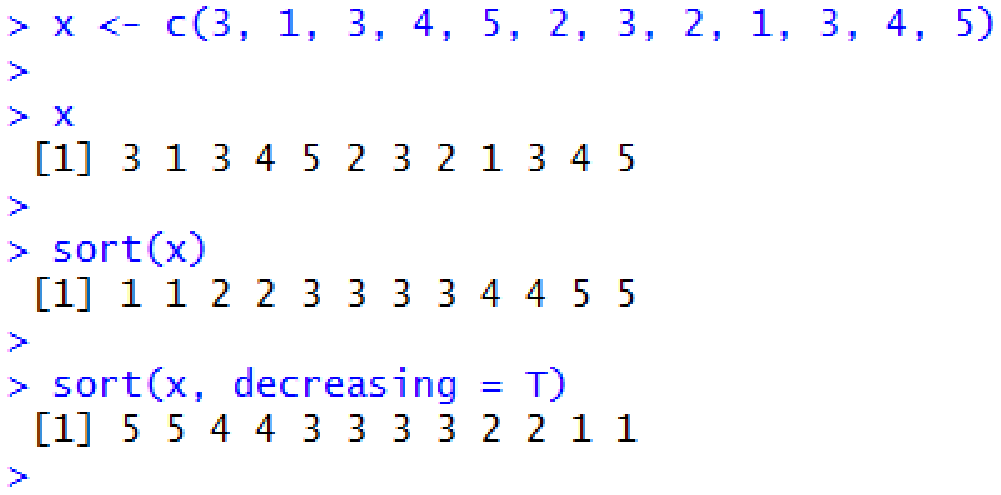
Exercise 12 (Solution)



```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y <- 5:15
> y
 [1]  5  6  7  8  9 10 11 12 13 14 15
> union(x,y)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> intersect(x, y)
[1]  5  6  7  8  9 10
> setdiff(x, y)
[1] 1 2 3 4
> setdiff(y, x)
[1] 11 12 13 14 15
> 2 %in% x
[1] TRUE
>
```

Ordering a vector

- The **sort**() function is used to sort the elements of a vector



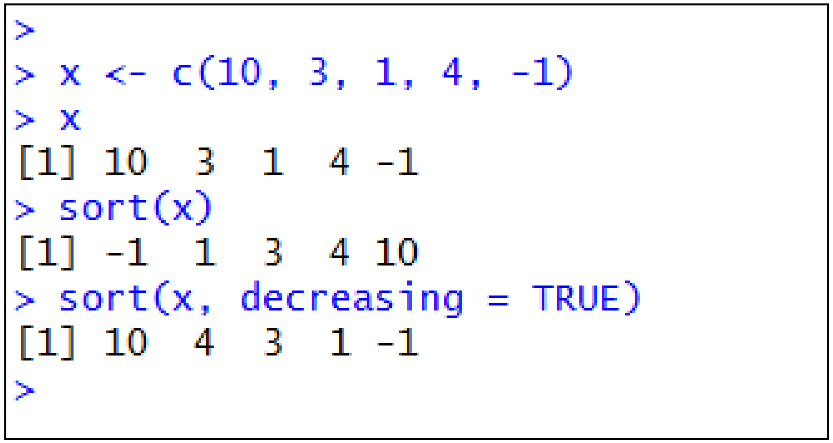
```
> x <- c(3, 1, 3, 4, 5, 2, 3, 2, 1, 3, 4, 5)
>
> x
 [1] 3 1 3 4 5 2 3 2 1 3 4 5
>
> sort(x)
 [1] 1 1 2 2 3 3 3 3 4 4 5 5
>
> sort(x, decreasing = T)
 [1] 5 5 4 4 3 3 3 3 2 2 1 1
>
```

Exercise 13

- Create a vector called "x" with the numbers 10, 3, 1, 4 and -1
- Sort in ascending order
- Sort in descending order

Exercise 13 (Solution)

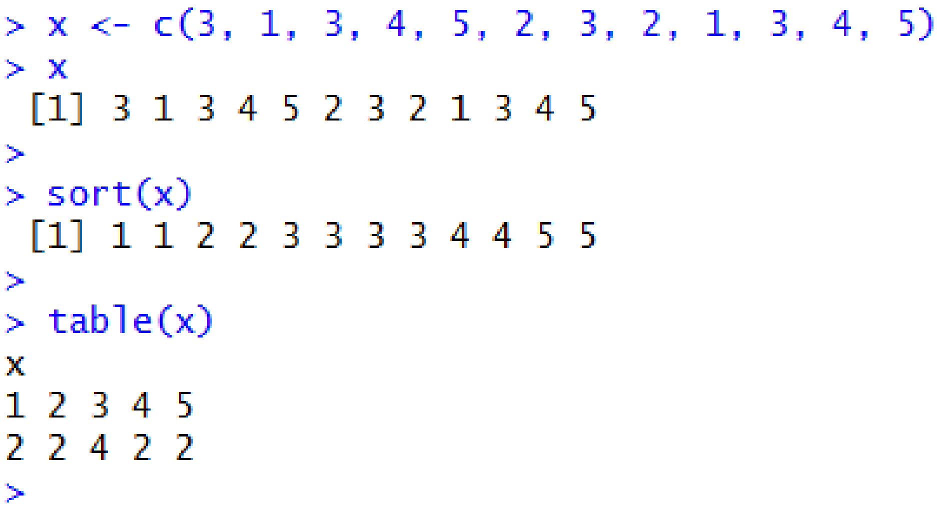
- Create a vector called "x" with the numbers 10, 3, 1, 4 and -1
- Sort in ascending order
- Sort in descending order



```
>
> x <- c(10, 3, 1, 4, -1)
> x
[1] 10  3  1  4 -1
> sort(x)
[1] -1  1  3  4 10
> sort(x, decreasing = TRUE)
[1] 10  4  3  1 -1
>
```

Counting elements in a vector

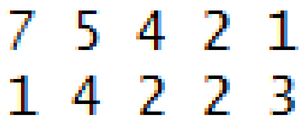
- The **table**() function is used to obtain the frequency of the elements of a vector



```
> x <- c(3, 1, 3, 4, 5, 2, 3, 2, 1, 3, 4, 5)
> x
 [1] 3 1 3 4 5 2 3 2 1 3 4 5
>
> sort(x)
 [1] 1 1 2 2 3 3 3 3 4 4 5 5
>
> table(x)
x
1 2 3 4 5
2 2 4 2 2
>
```

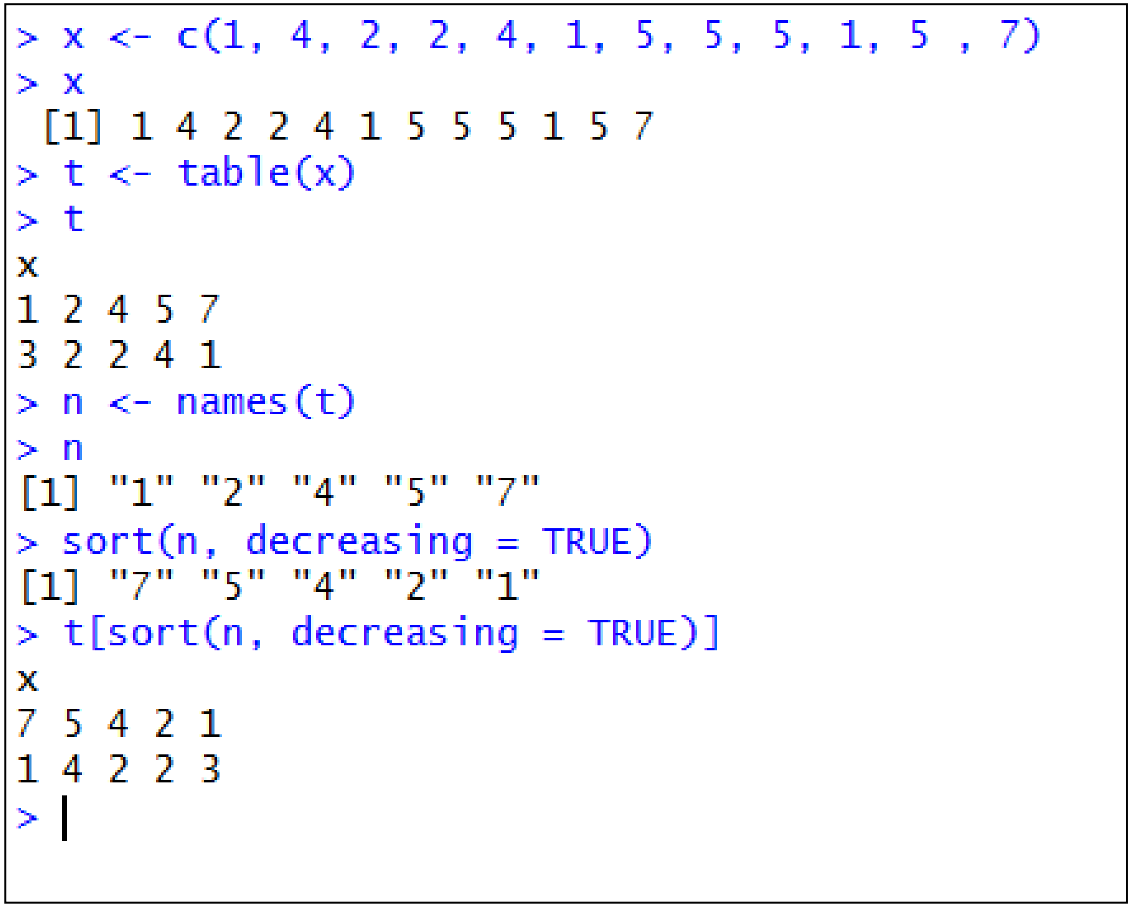
Exercise 14

- Create a vector "x" with elements 1, 4, 2, 2, 4, 1, 5, 5, 5, 1, 5 and 7
- Create a vector "t" with the frequency of the elements of the vector "x"
- Create a vector "n" with the names of the vector "t"
- Show vector "n" in descending order
- Displays the descending ordered table:



```
7 5 4 2 1
1 4 2 2 3
```

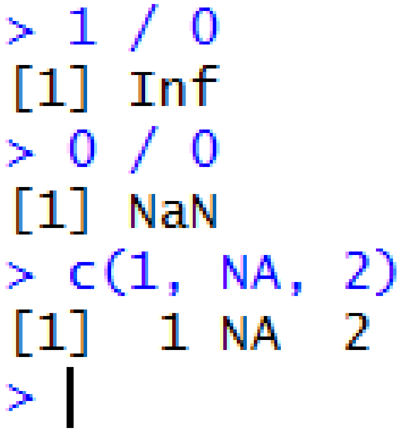
Exercise 14 (Solution)



```
> x <- c(1, 4, 2, 2, 4, 1, 5, 5, 5, 1, 5 , 7)
> x
 [1] 1 4 2 2 4 1 5 5 5 1 5 7
> t <- table(x)
> t
x
1 2 4 5 7
3 2 2 4 1
> n <- names(t)
> n
[1] "1" "2" "4" "5" "7"
> sort(n, decreasing = TRUE)
[1] "7" "5" "4" "2" "1"
> t[sort(n, decreasing = TRUE)]
x
7 5 4 2 1
1 4 2 2 3
> |
```

Special numbers

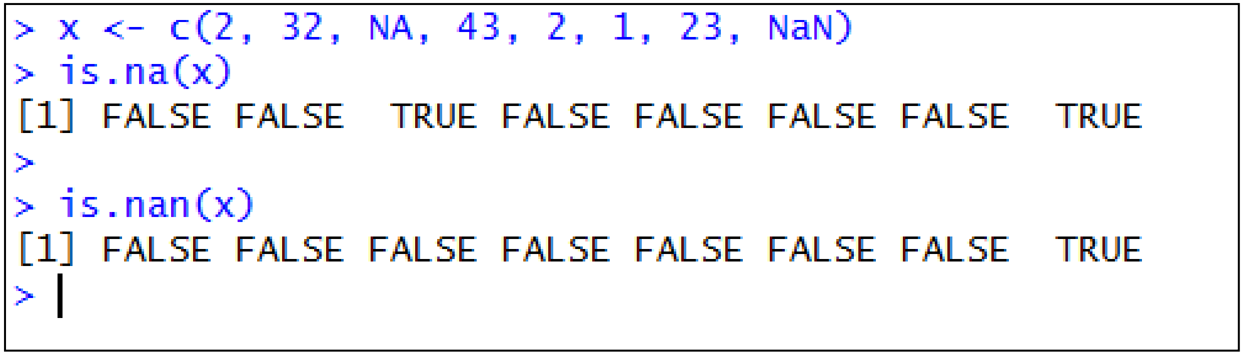
- **Inf** represents infinite / -**Inf** represents minus infinity
- **NaN** represents an undefined value (Not a Number)
- **NA** It represents a nonexistent value



```
> 1 / 0
[1] Inf
> 0 / 0
[1] NaN
> c(1, NA, 2)
[1]  1 NA  2
> 
```

Special numbers

- The **is.na ()** function is used to verify the data that is **NA** in a vector
- In the same way **is.nan ()** can be used for **NaN** data



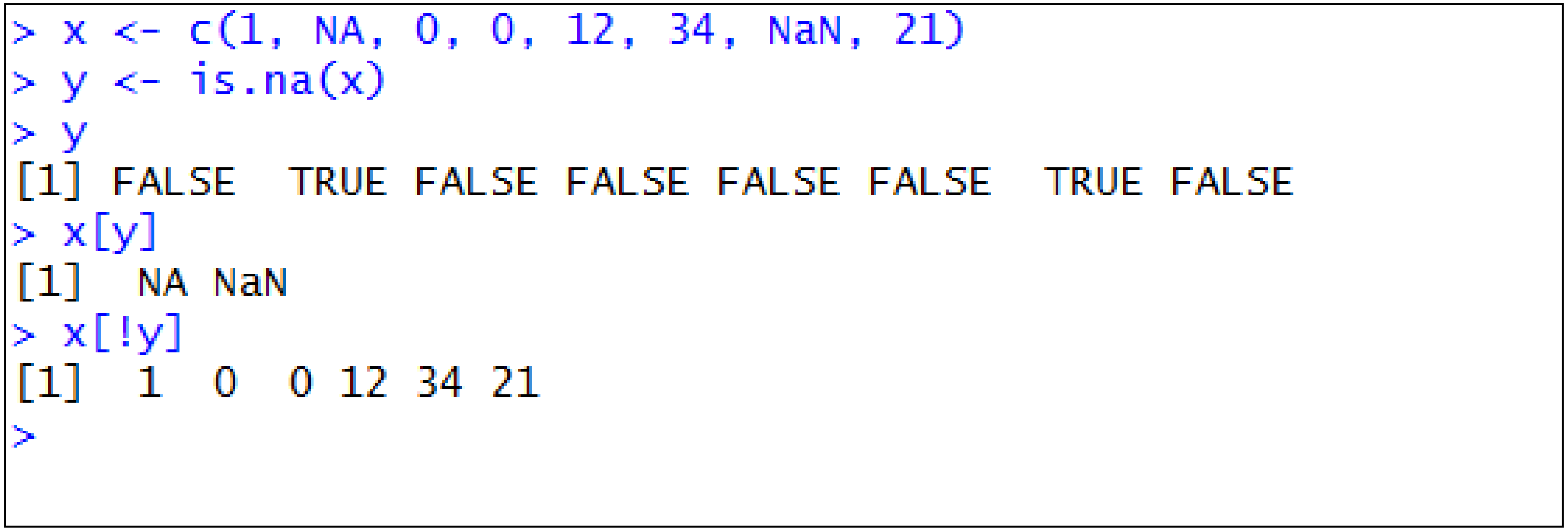
```
> x <- c(2, 32, NA, 43, 2, 1, 23, NaN)
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
>
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
> |
```

Exercise 15

- Create a vector called «x» with the following elements: 1, NA, 0, 0, 12, 34, NaN, 21
- Create a logical vector called "y" indicating that positions are not filled in the vector "x" (TRUE if the value is NA or NaN)
- Show values of "x" that are empty
- Show values of "x" that are filled (Not empty)

NOTE: Use the vector "y" to help you

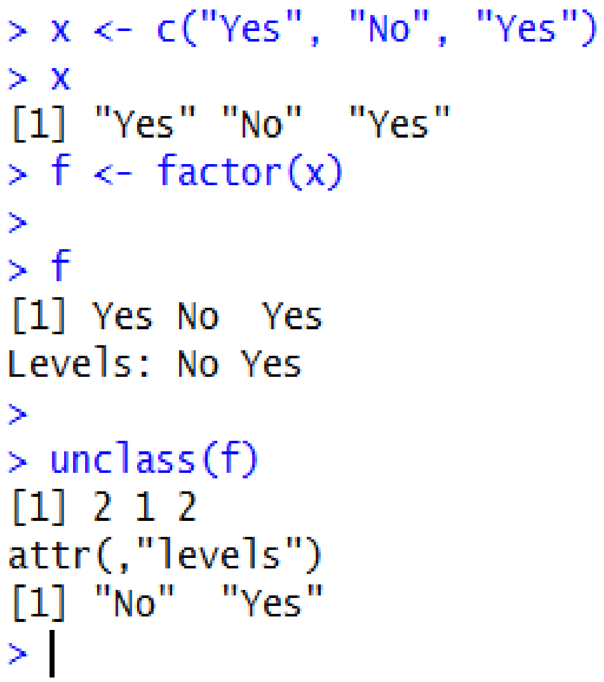
Exercise 15 (Solution)



```
> x <- c(1, NA, 0, 0, 12, 34, NaN, 21)
> y <- is.na(x)
> y
[1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
> x[y]
[1]  NA NaN
> x[!y]
[1]  1  0  0 12 34 21
>
```

Factors

- The factors are used to represent categories in R

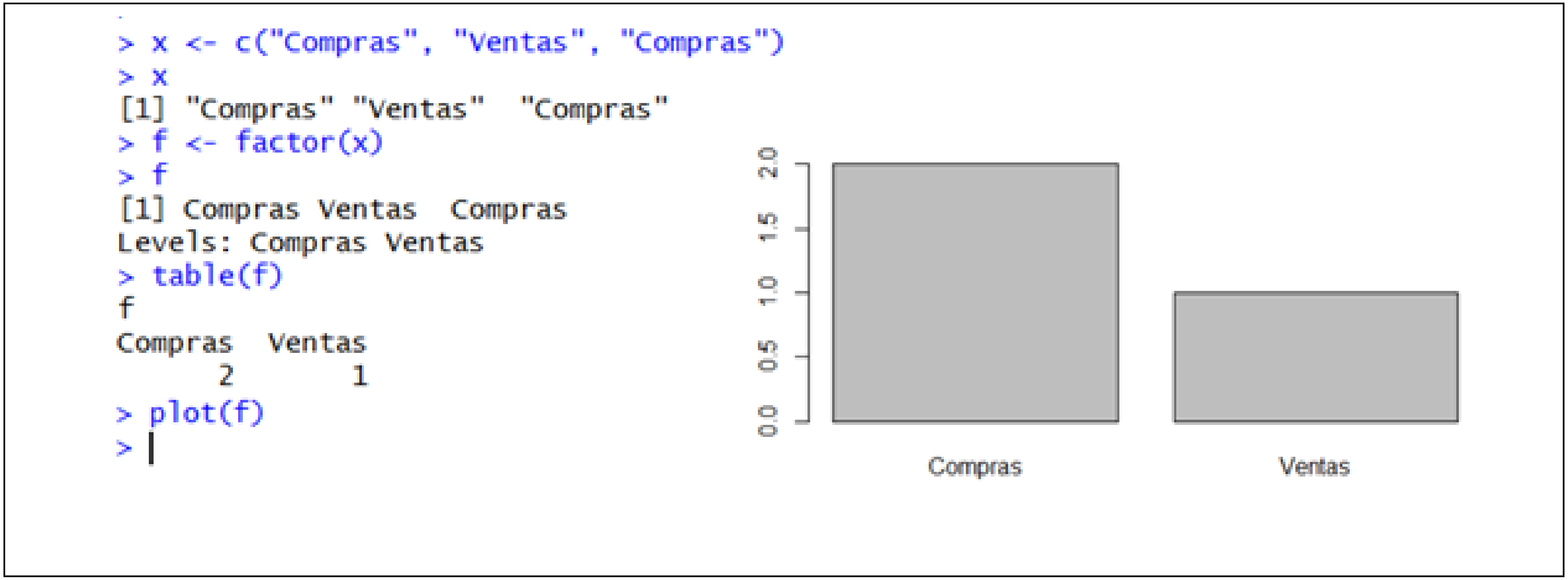


```
> x <- c("Yes", "No", "Yes")
> x
[1] "Yes" "No"  "Yes"
> f <- factor(x)
>
> f
[1] Yes No  Yes
Levels: No Yes
>
> unclass(f)
[1] 2 1 2
attr(,"levels")
[1] "No"  "Yes"
> 
```

Exercise 16

- Create a vector called "x" with the elements "Purchases", "Sales" and "Purchases"
- Create a factor called "f" with the content of "x"
- Print the frequency of the variable «f»
- Use the plot function show f graphically

Exercise 16 (Solution)



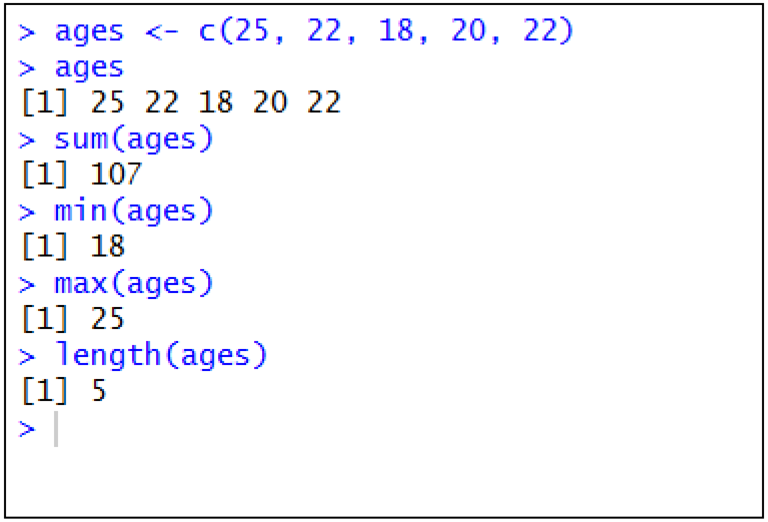
```
> x <- c("Compras", "Ventas", "Compras")
> x
[1] "Compras" "Ventas"  "Compras"
> f <- factor(x)
> f
[1] Compras Ventas  Compras
Levels: Compras Ventas
> table(f)
f
Compras  Ventas
      2       1
> plot(f)
> 
```

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Descriptive statistics

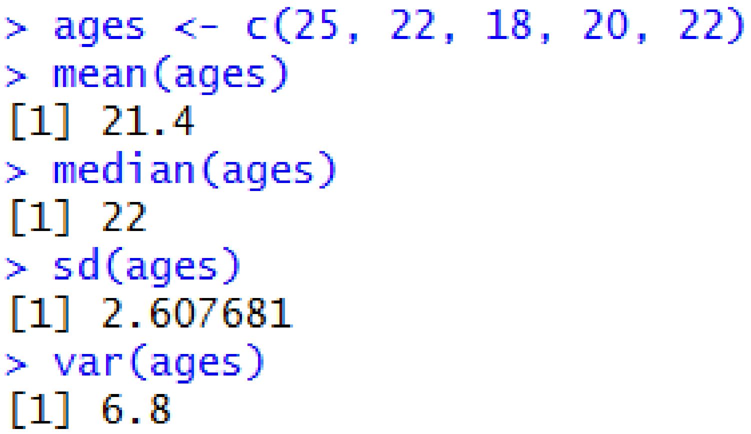
- **sum ()** returns the sum of all vector elements
- **length()** returns the length of a vector
- **min ()** the minimum value
- **max()** the maximum value



```
> ages <- c(25, 22, 18, 20, 22)
> ages
[1] 25 22 18 20 22
> sum(ages)
[1] 107
> min(ages)
[1] 18
> max(ages)
[1] 25
> length(ages)
[1] 5
> 
```

Descriptive statistics

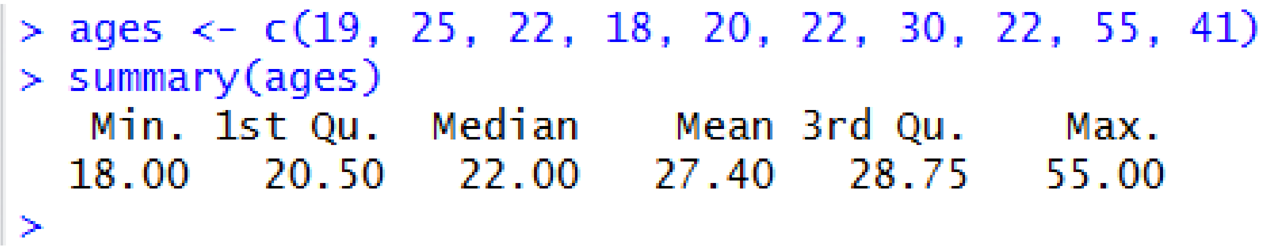
- **mean ()** returns the average of the values of a vector
- **median ()** the median
- **sd()** standard deviation
- **var()** variance



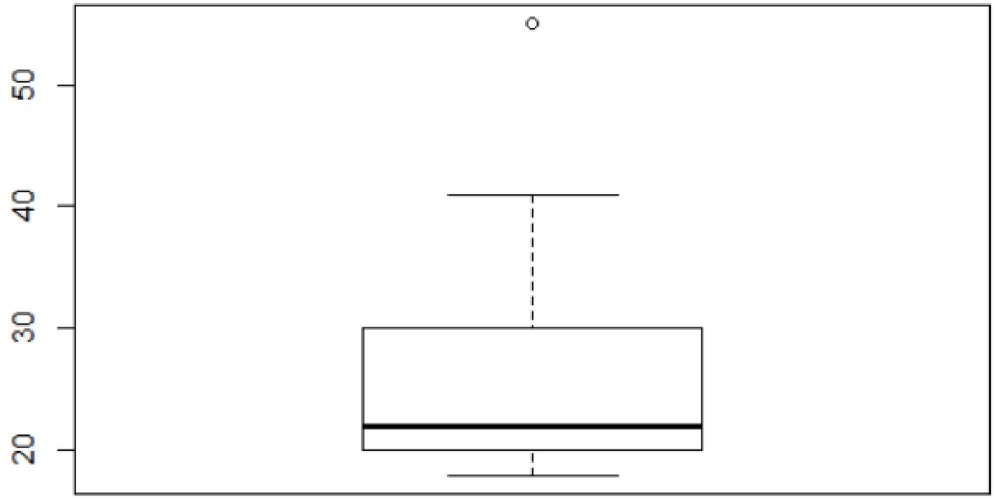
```
> ages <- c(25, 22, 18, 20, 22)
> mean(ages)
[1] 21.4
> median(ages)
[1] 22
> sd(ages)
[1] 2.607681
> var(ages)
[1] 6.8
```

Descriptive statistics

- **summary()** It shows data distribution



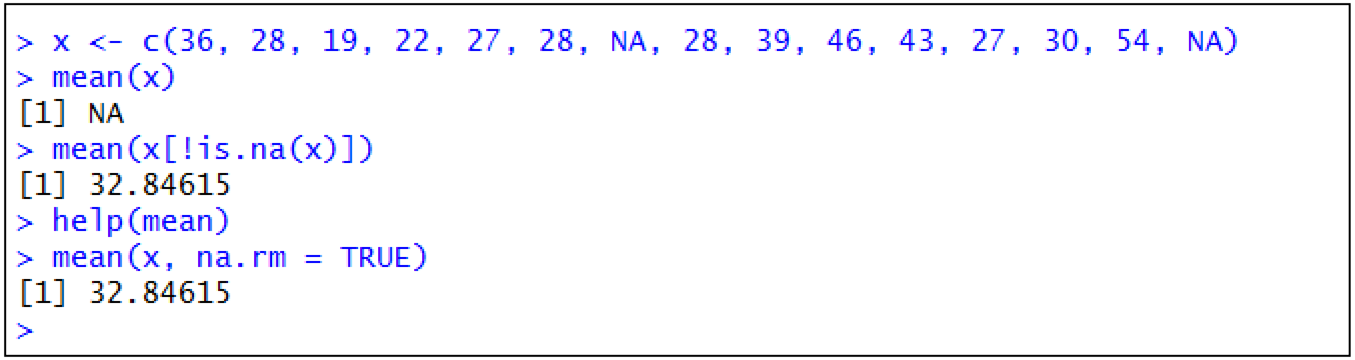
```
> ages <- c(19, 25, 22, 18, 20, 22, 30, 22, 55, 41)
> summary(ages)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.00   20.50   22.00   27.40   28.75   55.00 
> 
```



Exercise 17

- Creates a vector named "x" with the following values: 36, 28, 19, 22, 27, 28, NA, 28, 39, 46, 43, 27, 30, 54 and NA
- Calculate the average vector. What happen?
- Now calculate the mean using the function **mean** with function **is.na** to remove nulls
- Is it possible to calculate the average without using the is.na function?

Exercise 17 (Solution)



```
> x <- c(36, 28, 19, 22, 27, 28, NA, 28, 39, 46, 43, 27, 30, 54, NA)
> mean(x)
[1] NA
> mean(x[!is.na(x)])
[1] 32.84615
> help(mean)
> mean(x, na.rm = TRUE)
[1] 32.84615
>
```

Exercise 18

- Create a vector called "x" with the following values: 36, 28, 19, 22, 27, 28, NA, 28, 39, 46, 43, 27, 30, 54 and NA
- Prints the vector size
- Print your average (without using the mean function)
- Print your range (maximum value minus minimum)
- Prints its variance (without using the var function)

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

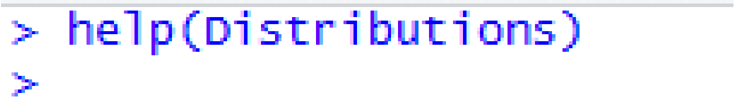
- Based on the previous calculation it prints the standard deviation
- Calculate the median (without using median)
- Calculate the mode (value that is most repeated from the vector

Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Probability distributions in R

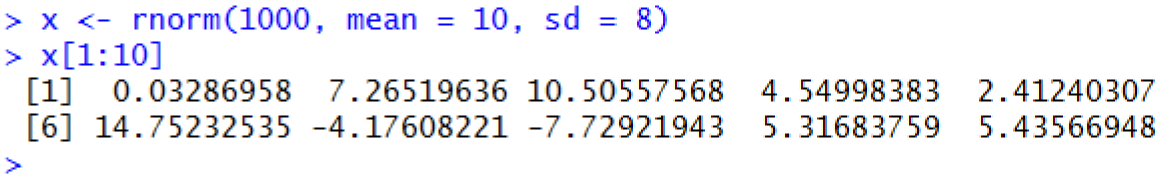
- R comes with a series of standard probability distributions, such as the normal distribution
- This allows us to generate random numbers according to a certain distribution
- Use the **help (distributions)** function to see them



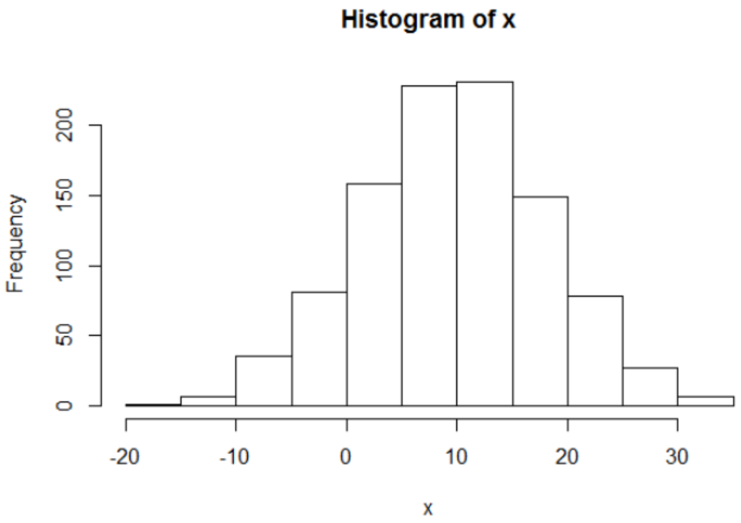
```
> help(Distributions)
>
```

Normal distribution

- To generate random numbers according to the normal distribution, use **rnorm()**

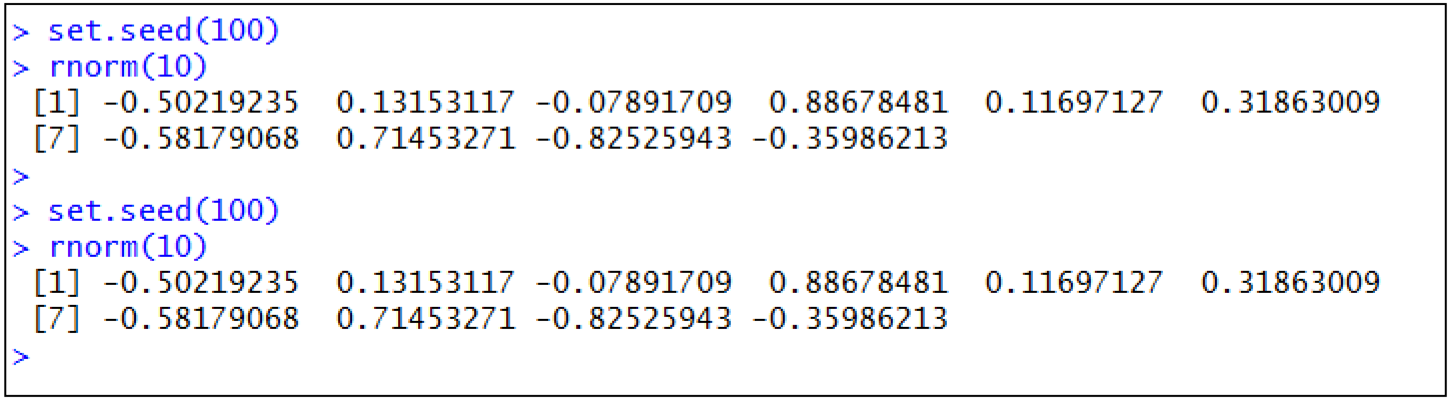


```
> x <- rnorm(1000, mean = 10, sd = 8)
> x[1:10]
 [1]  0.03286958  7.26519636 10.50557568  4.54998383  2.41240307
 [6] 14.75232535 -4.17608221 -7.72921943  5.31683759  5.43566948
>
```



Random seed

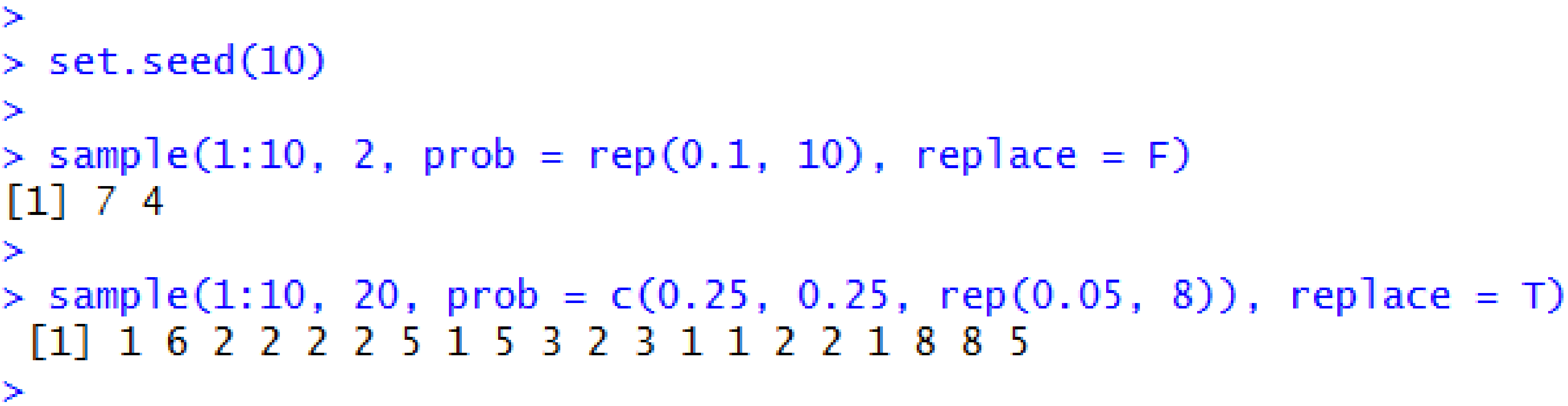
- The **set.seed()** function allows you to set the random seed for all functions that generate random values
- In this way you can get reproducible examples



```
> set.seed(100)
> rnorm(10)
 [1] -0.50219235  0.13153117 -0.07891709  0.88678481  0.11697127  0.31863009
 [7] -0.58179068  0.71453271 -0.82525943 -0.35986213
>
> set.seed(100)
> rnorm(10)
 [1] -0.50219235  0.13153117 -0.07891709  0.88678481  0.11697127  0.31863009
 [7] -0.58179068  0.71453271 -0.82525943 -0.35986213
>
```

Sampling

- The **sample ()** function is used to obtain an example of a vector of numbers

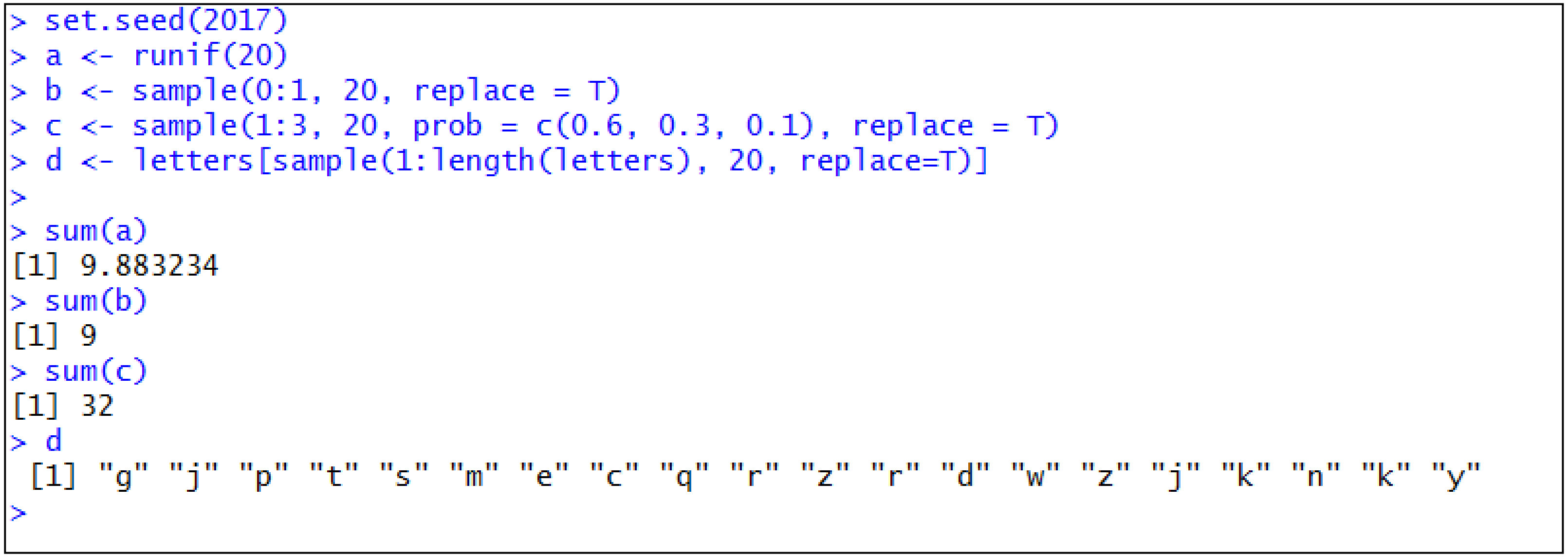


```
>
> set.seed(10)
>
> sample(1:10, 2, prob = rep(0.1, 10), replace = F)
[1] 7 4
>
> sample(1:10, 20, prob = c(0.25, 0.25, rep(0.05, 8)), replace = T)
 [1] 1 6 2 2 2 2 5 1 5 3 2 3 1 1 2 2 1 8 8 5
>
```

Exercise 19

- Set the random seed to 2017
- Create a vector called a of 20 random numbers according to the uniform distribution
- Create a vector called «b» of 20 random numbers with values between 0 and 1
- Create a vector called «c» of 20 random numbers between 1 and 3 where the probability of a 1 coming out is 60%, 2 is 30% and 3 is 10%
- Create a vector called “d” of 20 random letters based on the vector letters

Exercise 19 (Solution)



```
> set.seed(2017)
> a <- runif(20)
> b <- sample(0:1, 20, replace = T)
> c <- sample(1:3, 20, prob = c(0.6, 0.3, 0.1), replace = T)
> d <- letters[sample(1:length(letters), 20, replace=T)]
>
> sum(a)
[1] 9.883234
> sum(b)
[1] 9
> sum(c)
[1] 32
> d
 [1] "g" "j" "p" "t" "s" "m" "e" "c" "q" "r" "z" "r" "d" "w" "z" "j" "k" "n" "k" "y"
>
```

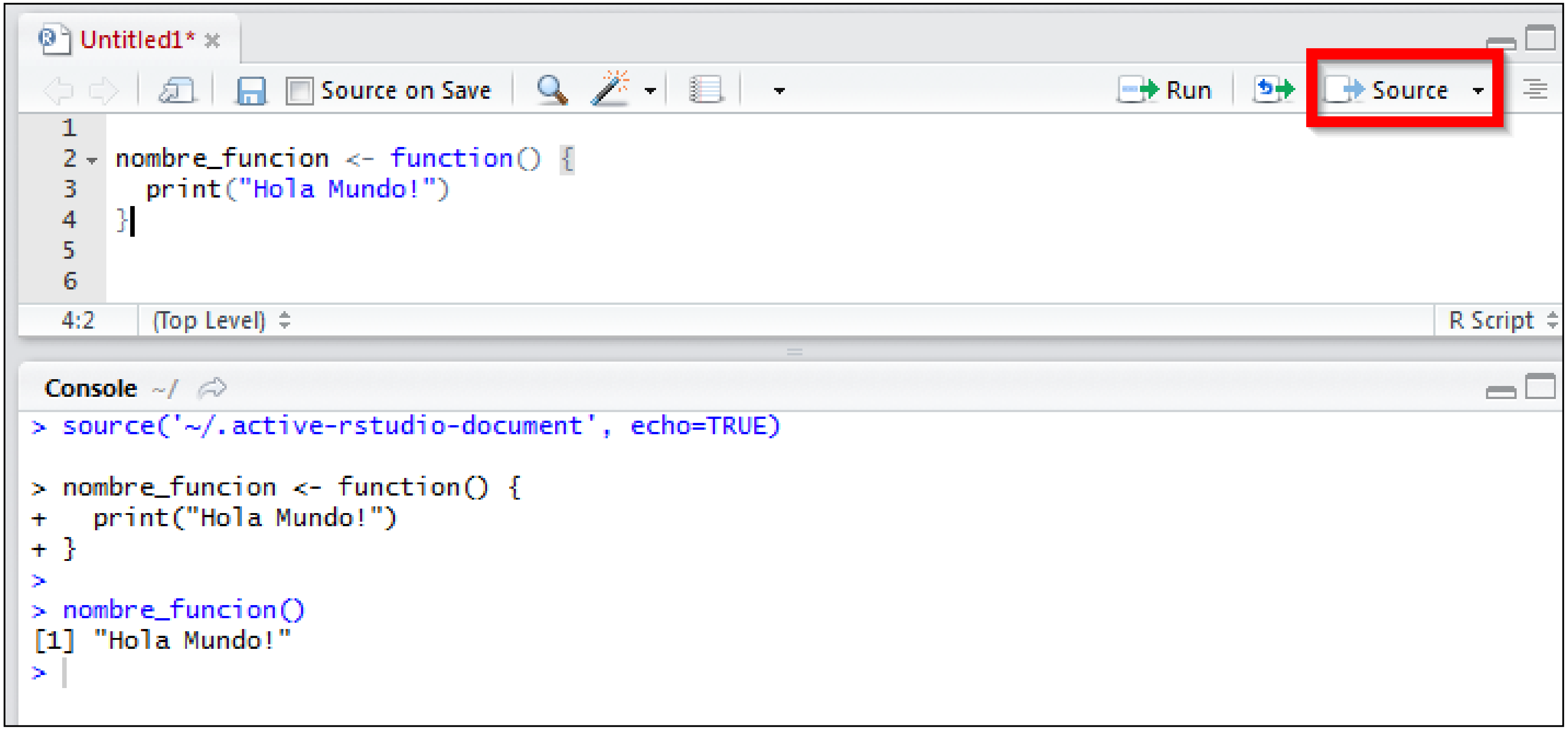
Index

- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Functions in R

- Everything you use in R is a function
- The libraries allow you to use additional features
- R can easily write new features

Functions in R

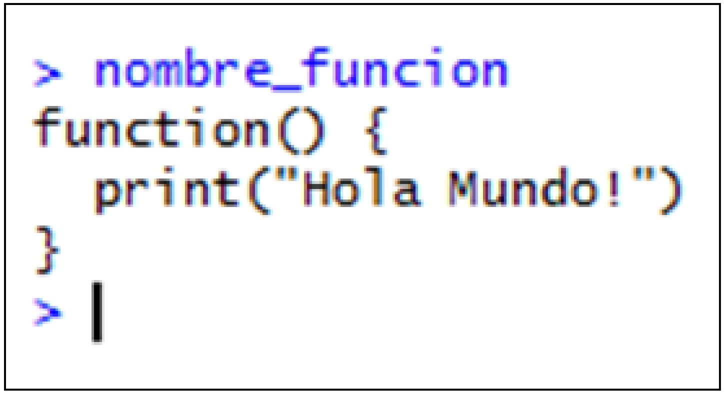


```
nombre_funcion <- function() {
  print("Hola Mundo!")
}
```

```
> source('~/.active-rstudio-document', echo=TRUE)

> nombre_funcion <- function() {
+   print("Hola Mundo!")
+ }
>
> nombre_funcion()
[1] "Hola Mundo!"
>
```

Functions in R

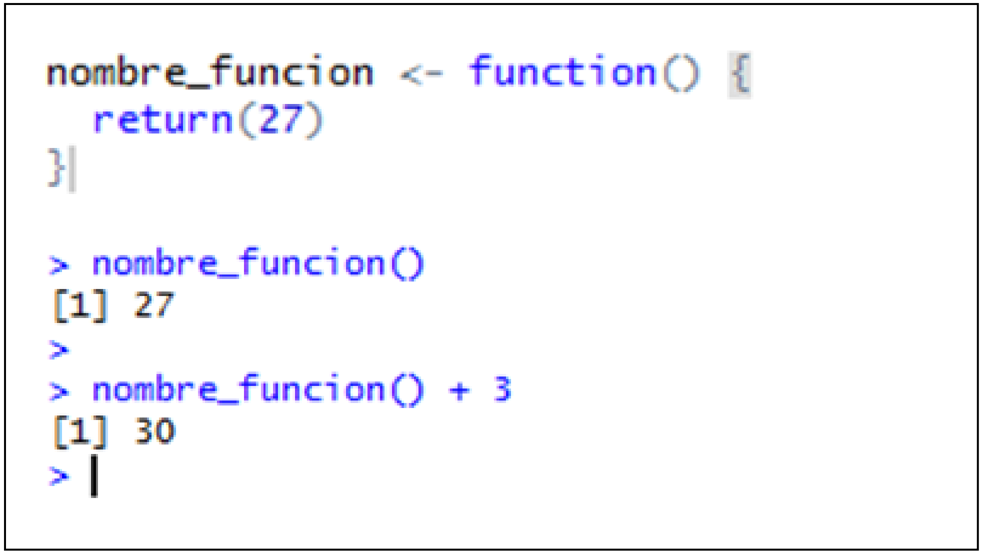
- When writing the code of a function without parentheses in the console, the function code is shown



```
> nombre_funcion
function() {
  print("Hola Mundo!")
}
> |
```

Functions in R

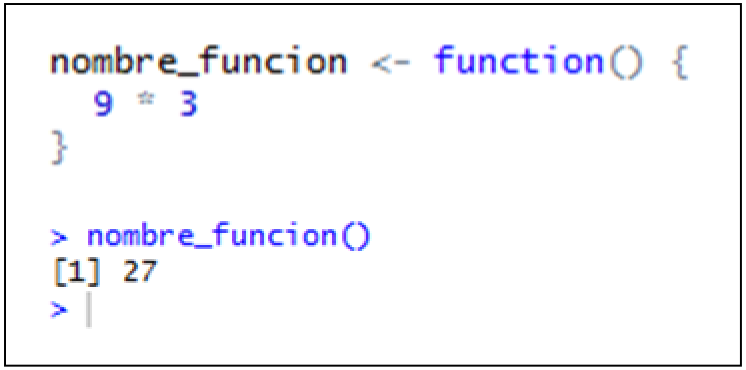
- The function **return**() allows the function to return a value



```
nombre_funcion <- function() {
  return(27)
}

> nombre_funcion()
[1] 27
>
> nombre_funcion() + 3
[1] 30
> 
```

Functions in R

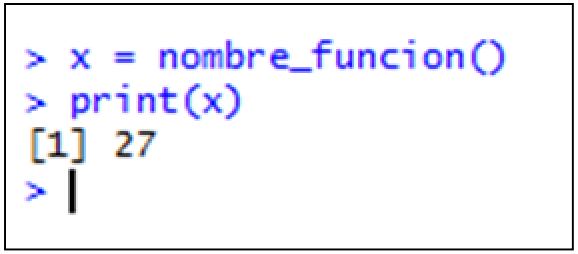
- In the case of no use of **return**() is always returned the result of the last expression



```
nombre_funcion <- function() {
  9 * 3
}

> nombre_funcion()
[1] 27
> 
```

Functions in R

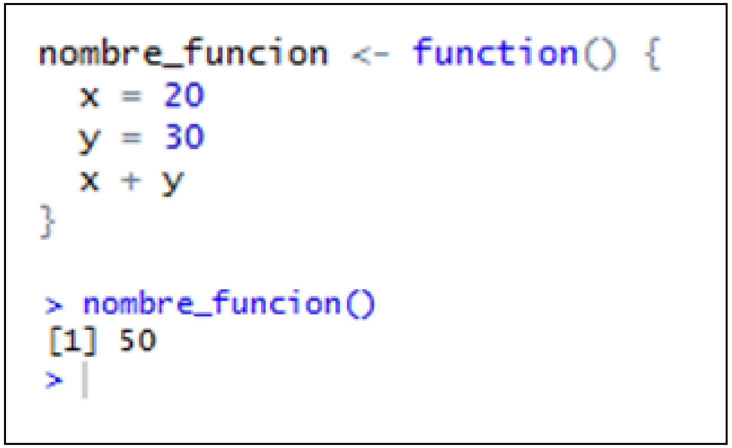
- The function returns a value that can be stored in a new variable



```
> x = nombre_funcion()
> print(x)
[1] 27
> |
```

Functions in R

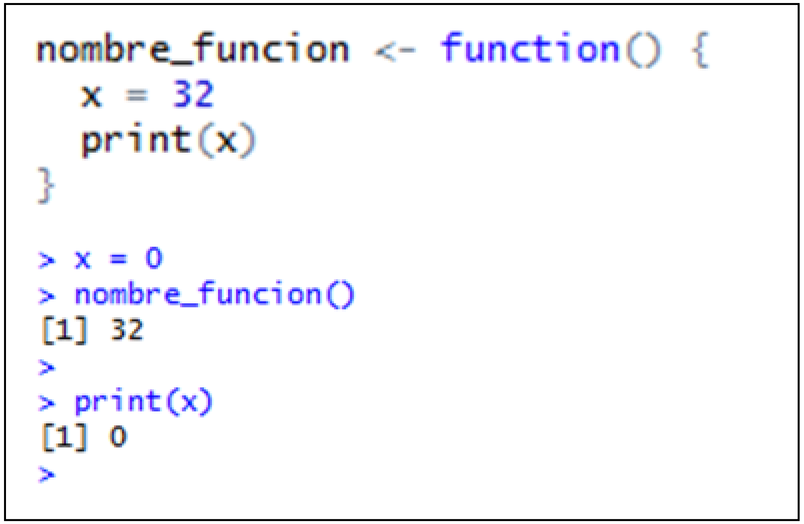
- Within a function we can use new variables ...



```
nombre_funcion <- function() {
   x = 20
   y = 30
   x + y
}

> nombre_funcion()
[1] 50
> 
```

Functions in R

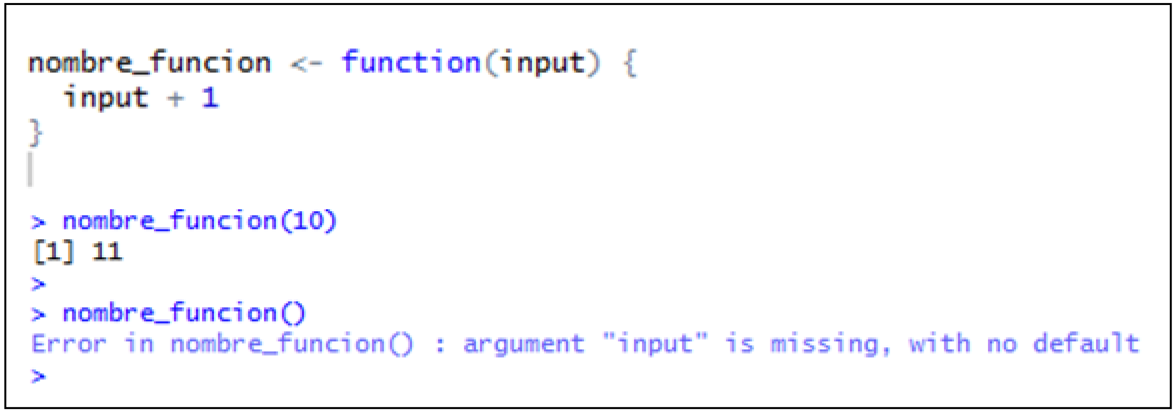
- We can even use a variable that has been previously declared without affecting its value



```
nombre_funcion <- function() {
  x = 32
  print(x)
}

> x = 0
> nombre_funcion()
[1] 32
>
> print(x)
[1] 0
>
```

Parameters

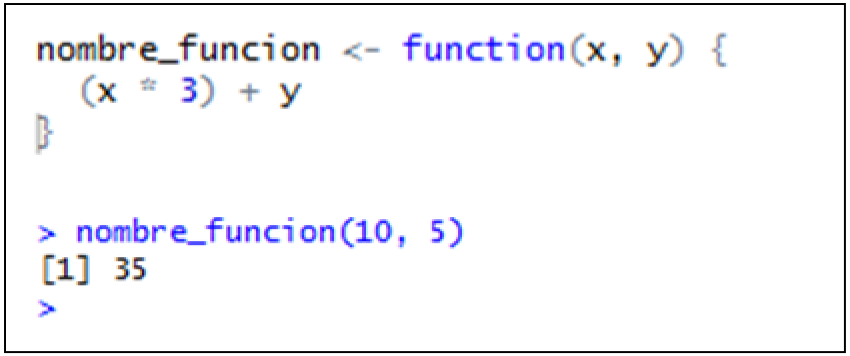
- To specify the input data to a function used parameters
- Note: You could use any variable name



```
nombre_funcion <- function(input) {
  input + 1
}

> nombre_funcion(10)
[1] 11
>
> nombre_funcion()
Error in nombre_funcion() : argument "input" is missing, with no default
>
```

Parameters

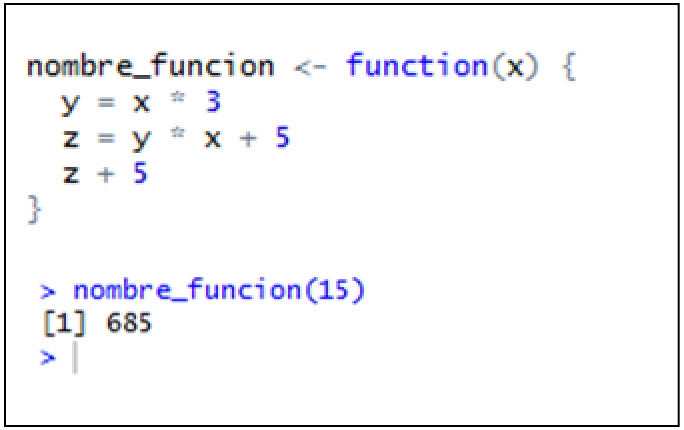
- We can use any number of parameters ...



```
nombre_funcion <- function(x, y) {
  (x * 3) + y
}


> nombre_funcion(10, 5)
[1] 35
>
```

Solving problems

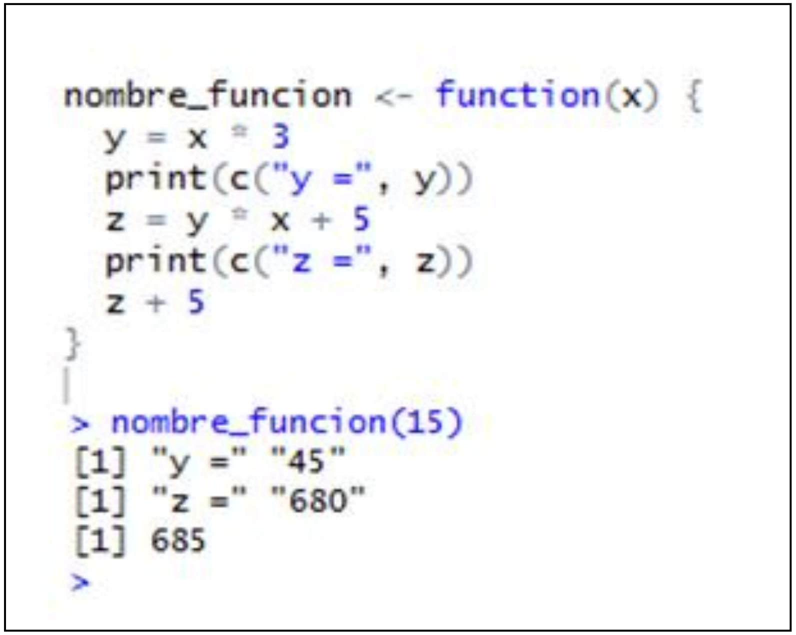
- A common tactic in case of problems is to print intermediate results



```
nombre_funcion <- function(x) {
  y = x * 3
  z = y * x + 5
  z + 5
}

 > nombre_funcion(15)
 [1] 685
 > |
```

Solving problems

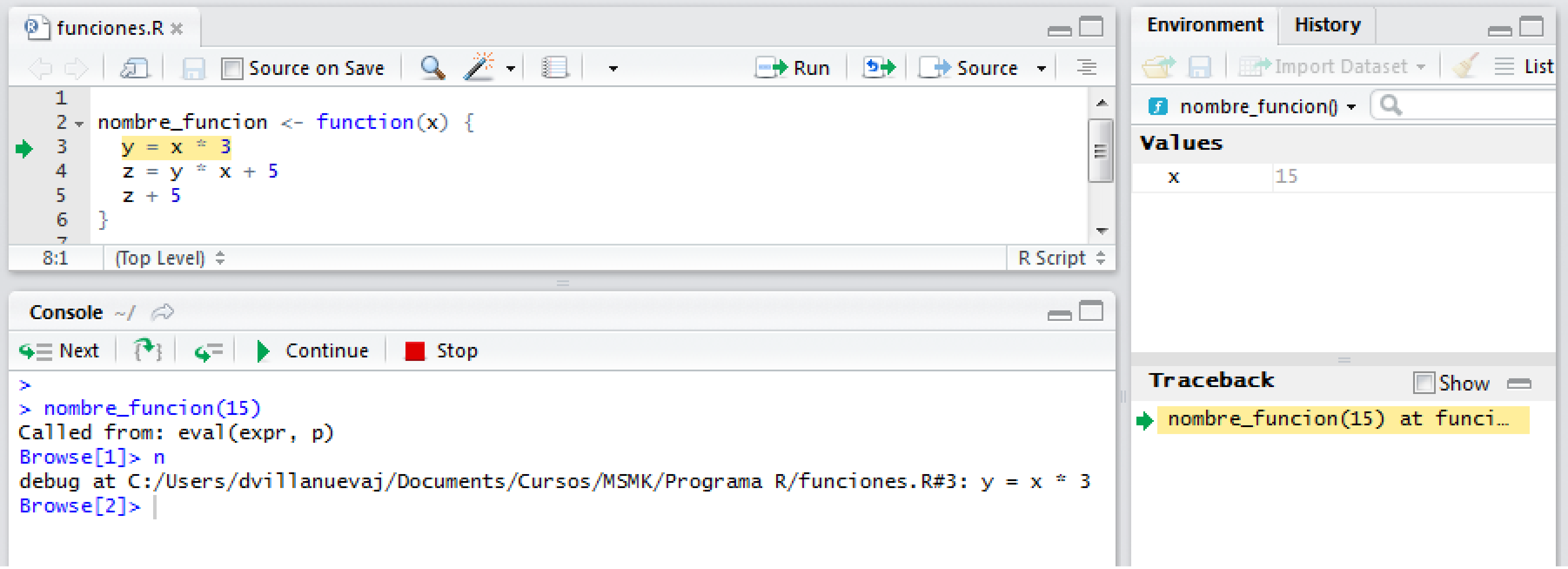
- A common tactic in case of problems is to print intermediate results



```
nombre_funcion <- function(x) {
  y = x * 3
  print(c("y =", y))
  z = y * x + 5
  print(c("z =", z))
  z + 5
}

> nombre_funcion(15)
[1] "y =" "45"
[1] "z =" "680"
[1] 685
>
```

Solving problems

- Another tactic is to set breakpoints and do "debug“ inside the function



```
nombre_funcion <- function(x) {
  y = x * 3
  z = y * x + 5
  z + 5
}
```

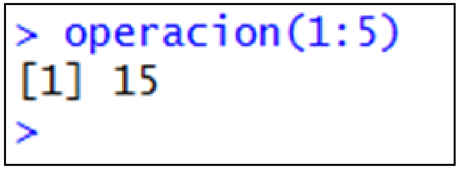
Values	
x	15

```
>
> nombre_funcion(15)
Called from: eval(expr, p)
Browse[1]> n
debug at C:/Users/dvillanuevaj/Documents/Cursos/MSMK/Programa R/funciones.R#3: y = x * 3
Browse[2]>
```

Traceback: nombre_funcion(15) at funci…

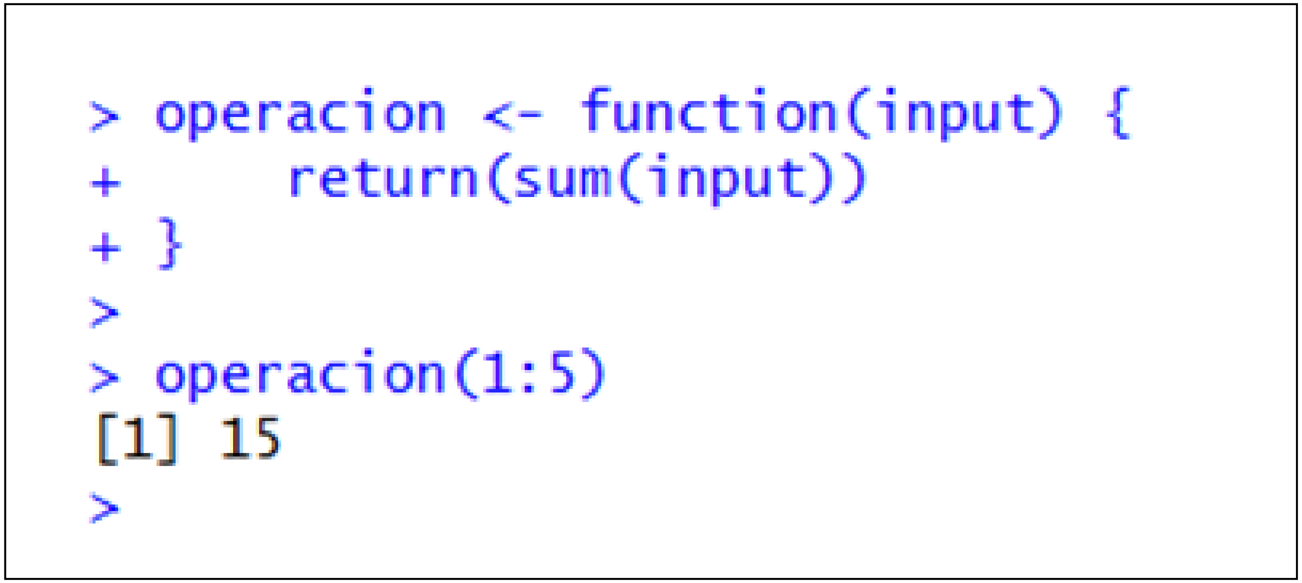
Exercise 20

- Write a function called "**operation**“ that accepts a single numeric parameter
- The purpose of the function is to add all the vector values
- Test function with the vector [1, 2, 3, 4, 5]



```
> operacion(1:5)
[1] 15
>
```

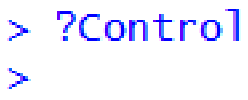
Exercise 20 (Solution)



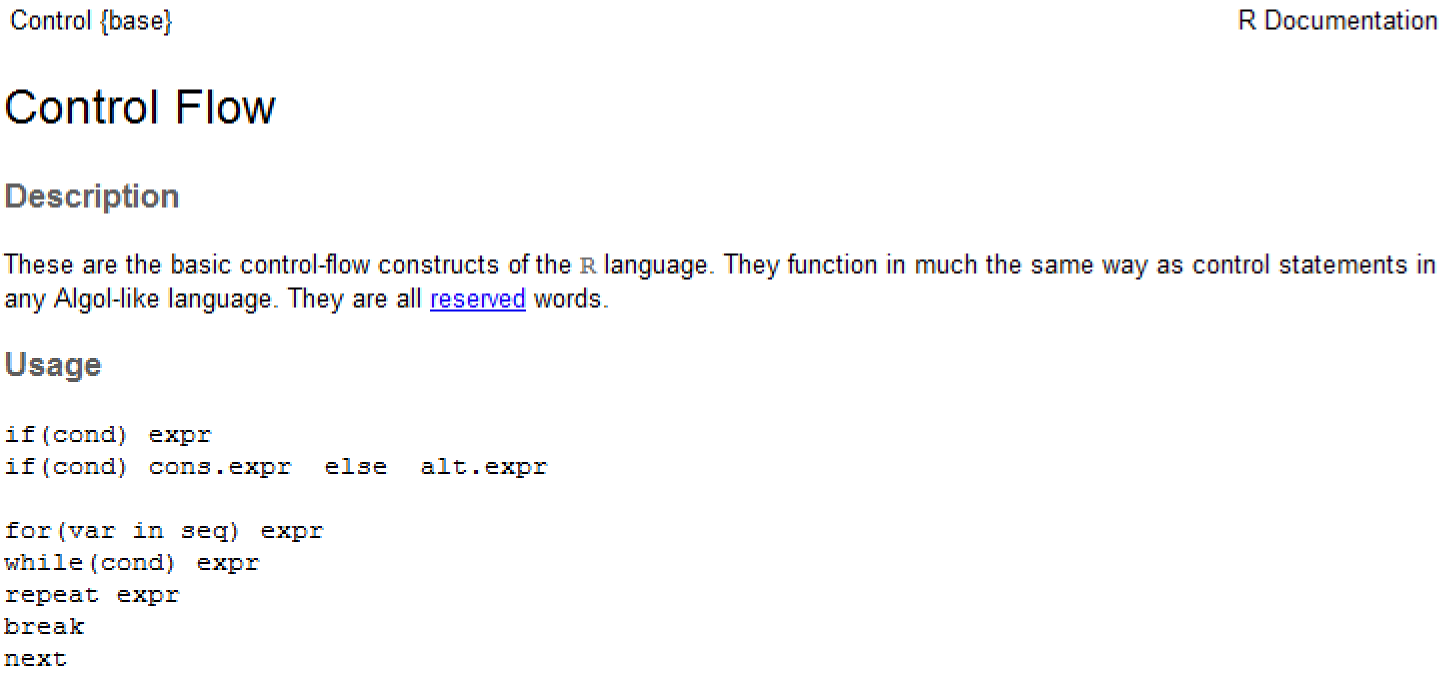
```
> operacion <- function(input) {
+     return(sum(input))
+ }
>
> operacion(1:5)
[1] 15
>
```

Control Structures

- Control structures allow control of command execution



```
> ?Control
>
```



Control {base}　　　　R Documentation

Control Flow

Description

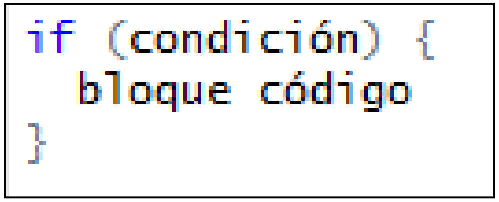
These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language. They are all reserved words.

Usage

```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

Control Structures

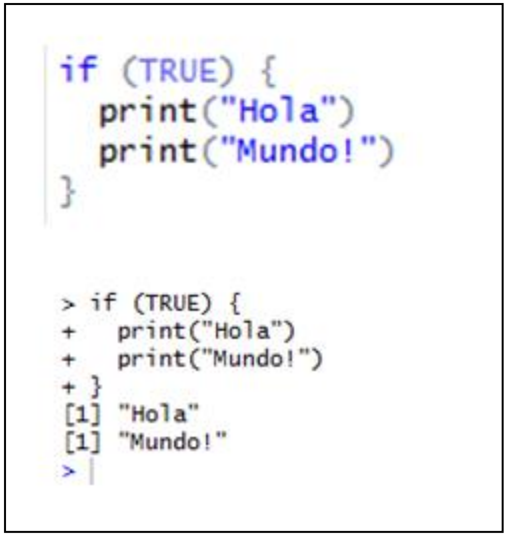
- **If()** allows only run a block of code if a condition is met



```
if (condición) {
  bloque código
}
```

Control Structures

- **If()** allows only run a block of code if a condition is met

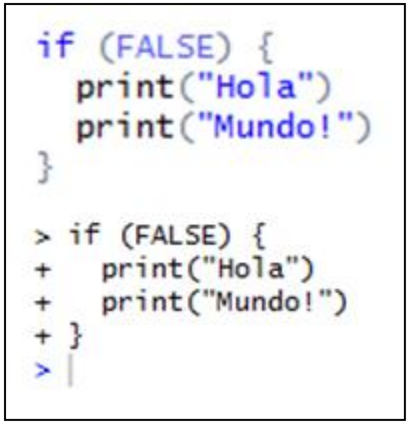


```
if (TRUE) {
  print("Hola")
  print("Mundo!")
}
```

```
> if (TRUE) {
+   print("Hola")
+   print("Mundo!")
+ }
[1] "Hola"
[1] "Mundo!"
> 
```

Control Structures

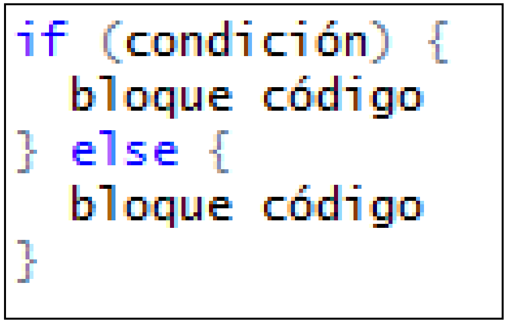
- **If()** allows only run a block of code if a condition is met



```
if (FALSE) {
  print("Hola")
  print("Mundo!")
}

> if (FALSE) {
+   print("Hola")
+   print("Mundo!")
+ }
> 
```

Control Structures

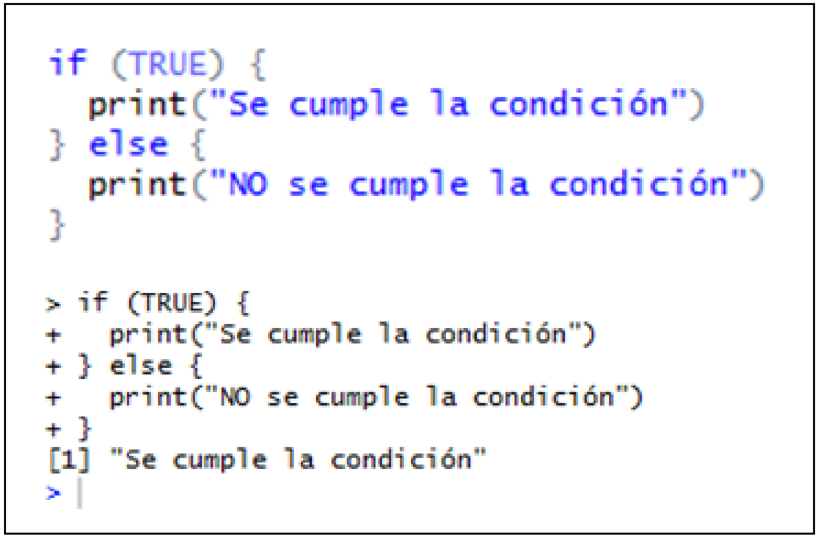
- **If()** can include a block **else** to run a block of code if the condition is not met



```
if (condición) {
  bloque código
} else {
  bloque código
}
```

Control Structures

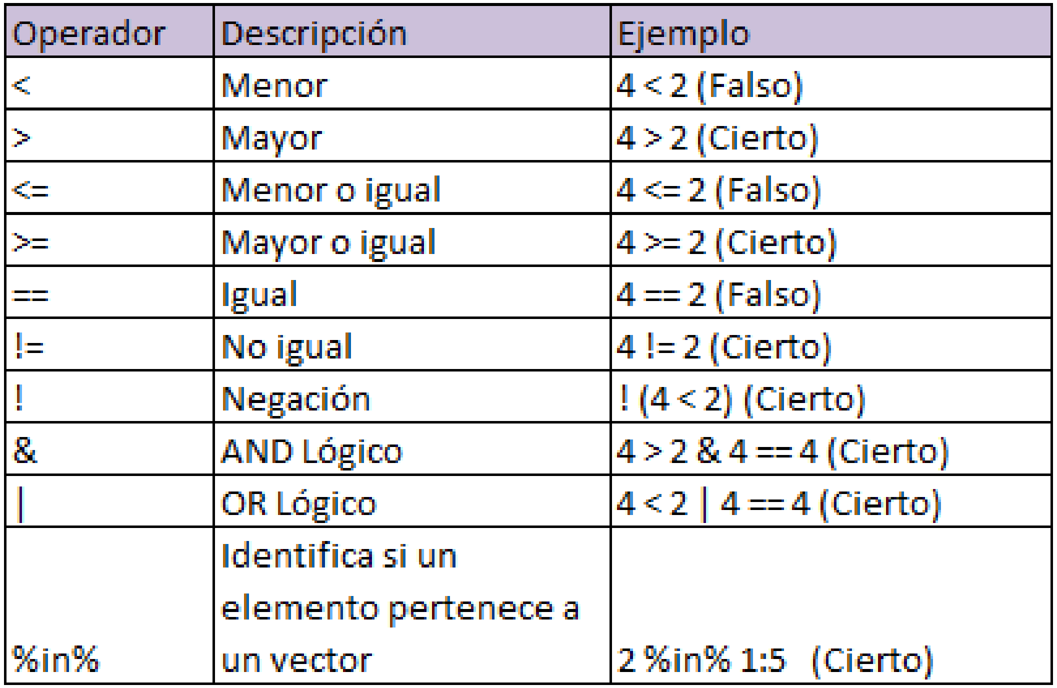
- **If()** can include a block **else** to run a block of code if the condition is not met



```
if (TRUE) {
  print("Se cumple la condición")
} else {
  print("NO se cumple la condición")
}

> if (TRUE) {
+   print("Se cumple la condición")
+ } else {
+   print("NO se cumple la condición")
+ }
[1] "Se cumple la condición"
> |
```

Logical operators

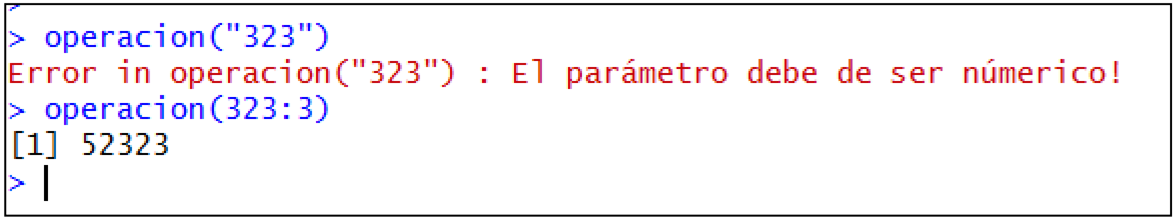
- R has the following logical operators



Operador	Descripción	Ejemplo
<	Menor	4 < 2 (Falso)
>	Mayor	4 > 2 (Cierto)
<=	Menor o igual	4 <= 2 (Falso)
>=	Mayor o igual	4 >= 2 (Cierto)
==	Igual	4 == 2 (Falso)
!=	No igual	4 != 2 (Cierto)
!	Negación	! (4 < 2) (Cierto)
&	AND Lógico	4 > 2 & 4 == 4 (Cierto)
|	OR Lógico	4 < 2 | 4 == 4 (Cierto)
%in%	Identifica si un elemento pertenece a un vector	2 %in% 1:5 (Cierto)

Exercise 21

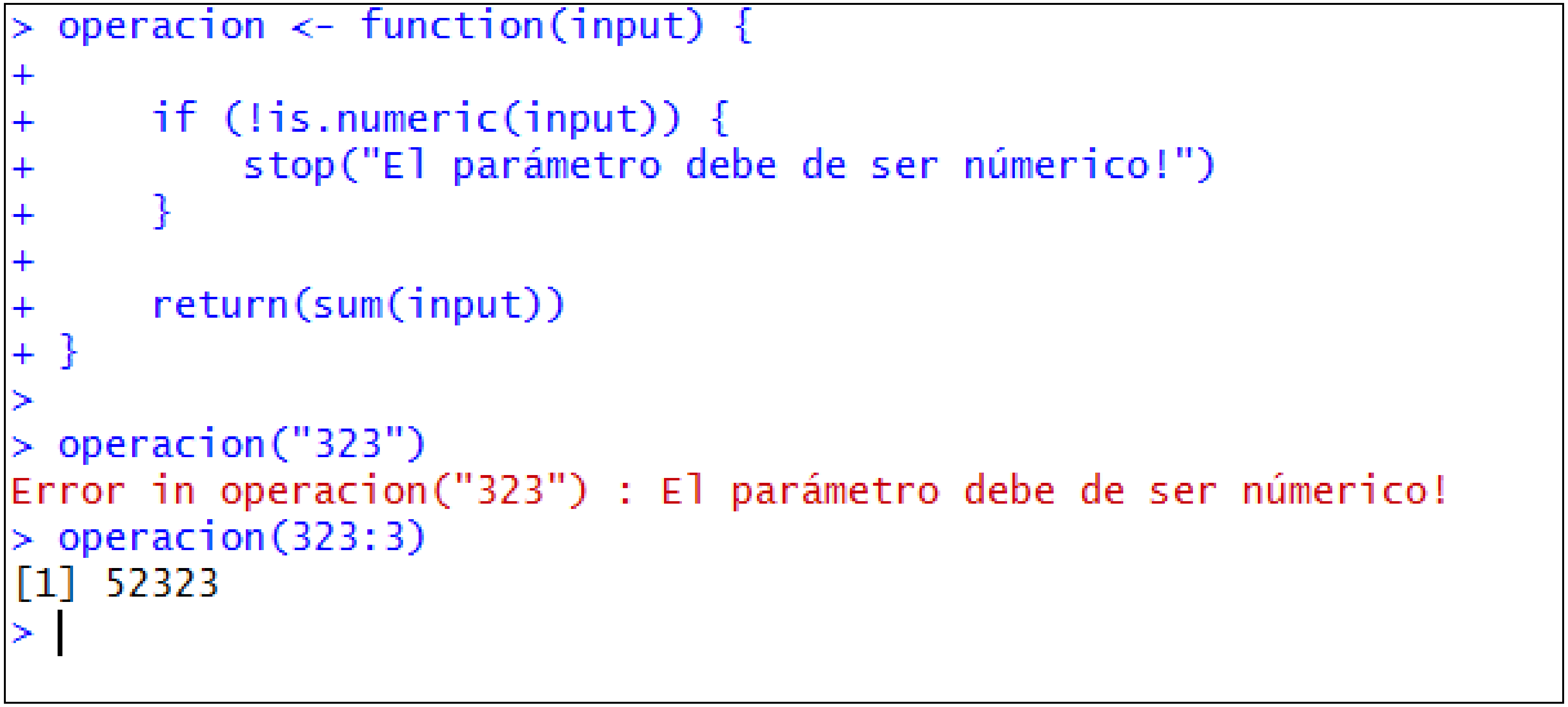
- Modifies the function “**operation**” to check the type of its argument
- For other value than a number, you must display the following error "parameter must be numeric!"
  - Use the **stop ()** function
- In the event that the argument is numeric must show the sum of its parts



```
> operacion("323")
Error in operacion("323") : El parámetro debe de ser númerico!
> operacion(323:3)
[1] 52323
> |
```

is.numeric()

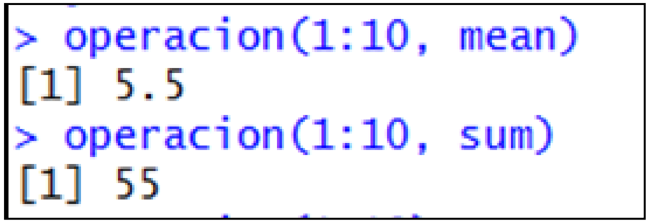
Exercise 21 (Solution)



```
> operacion <- function(input) {
+ 
+     if (!is.numeric(input)) {
+         stop("El parámetro debe de ser númerico!")
+     }
+ 
+     return(sum(input))
+ }
> 
> operacion("323")
Error in operacion("323") : El parámetro debe de ser númerico!
> operacion(323:3)
[1] 52323
> 
```

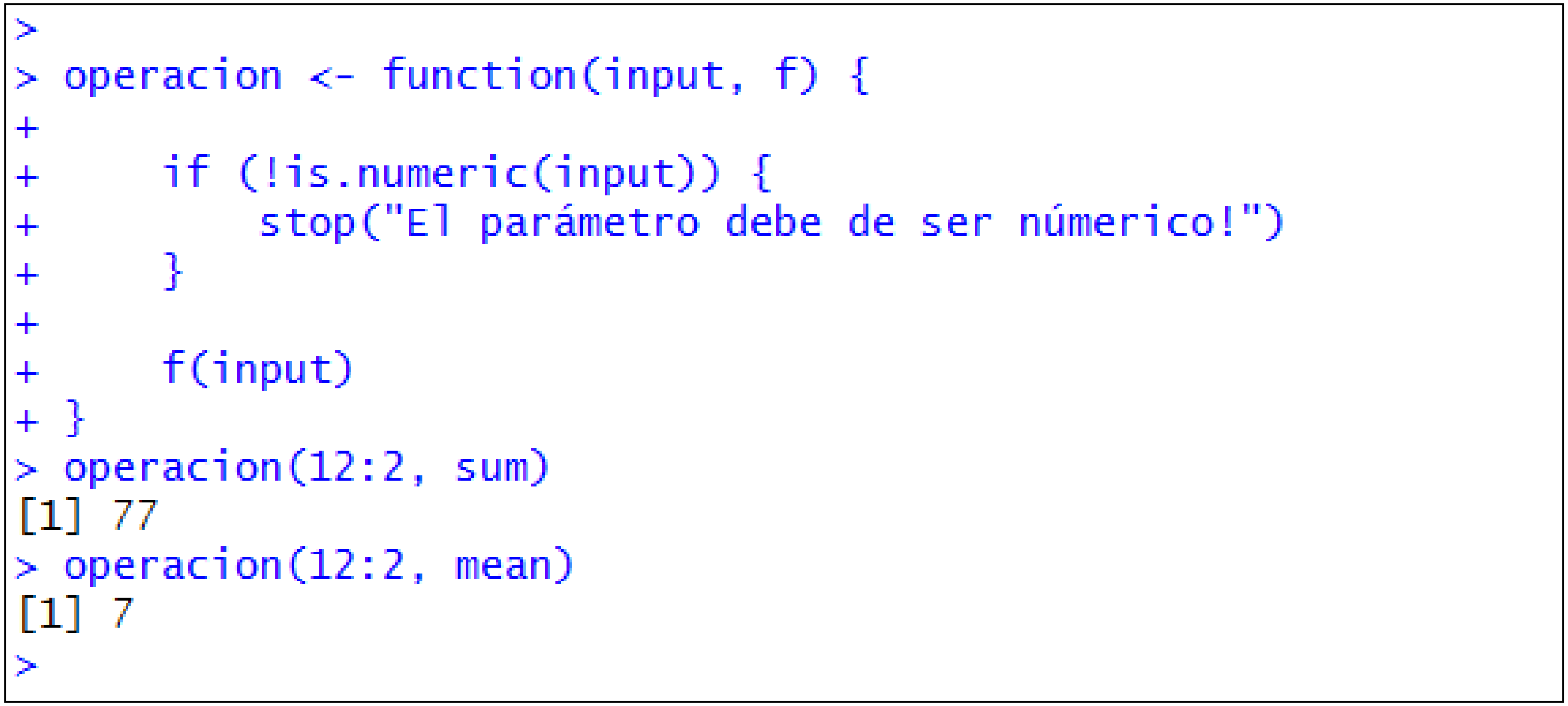
Exercise 22

- Modifies the function "**operation**" to allow the function to accept an argument that performs the operation



```
> operacion(1:10, mean)
[1] 5.5
> operacion(1:10, sum)
[1] 55
```

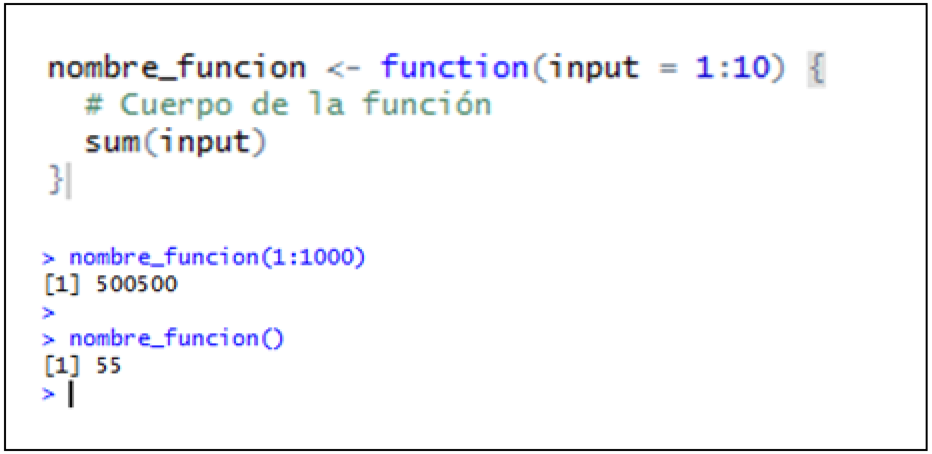
Exercise 22 (Solution)



```
>
> operacion <- function(input, f) {
+
+     if (!is.numeric(input)) {
+         stop("El parámetro debe de ser númerico!")
+     }
+
+     f(input)
+ }
> operacion(12:2, sum)
[1] 77
> operacion(12:2, mean)
[1] 7
>
```

Parameters with default values

- In a function, when a parameter is declared, you can specify a default value

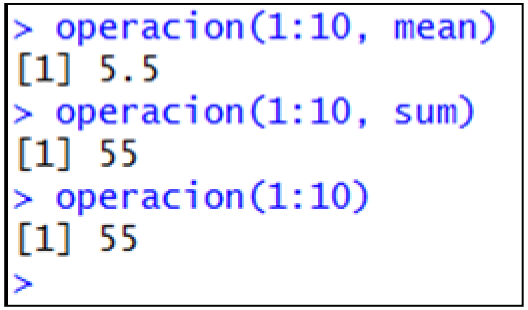


```
nombre_funcion <- function(input = 1:10) {
  # Cuerpo de la función
  sum(input)
}
```

```
> nombre_funcion(1:1000)
[1] 500500
>
> nombre_funcion()
[1] 55
> 
```

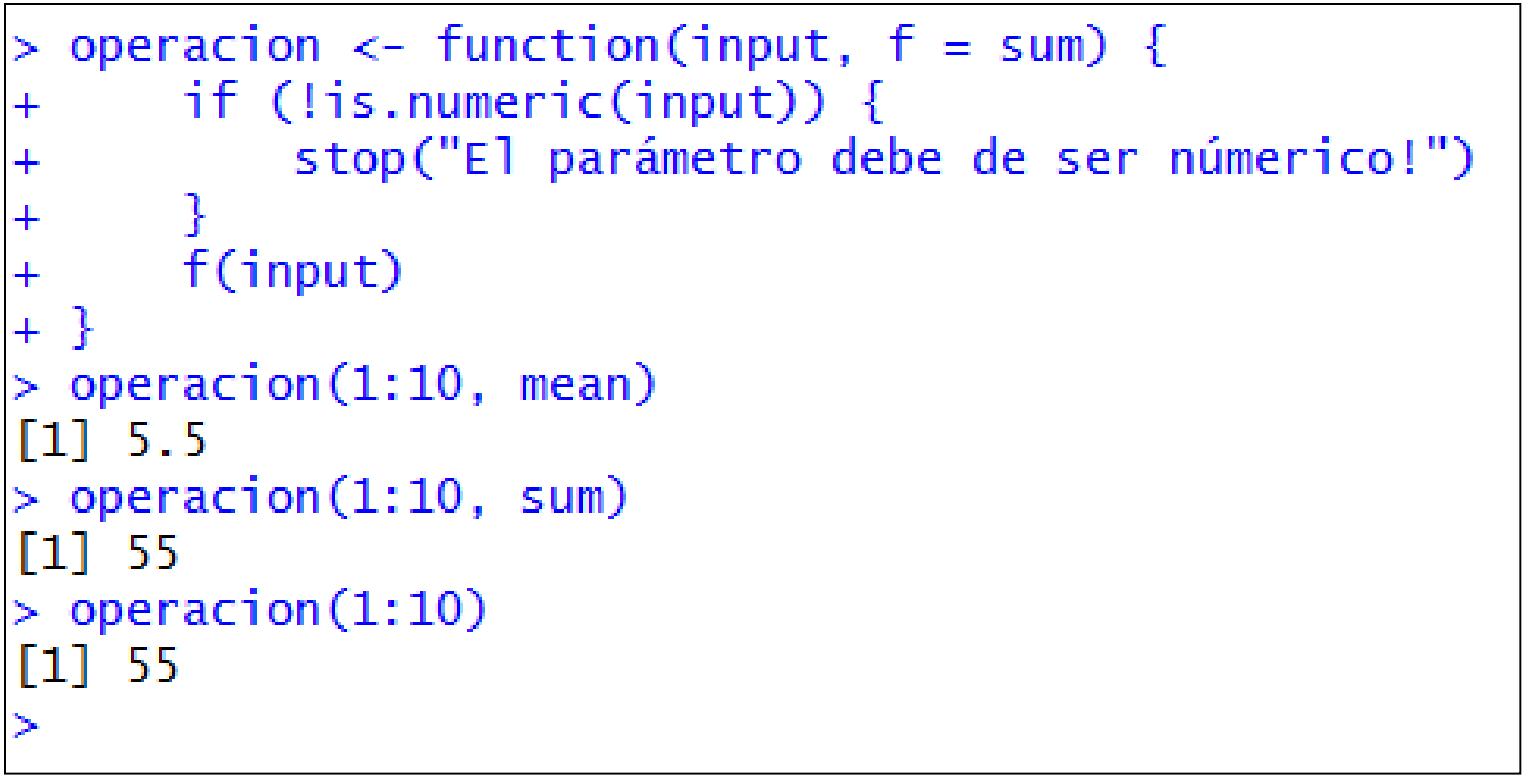
Exercise 23

- Modifies the behavior of the function “**operation**” so if it is not supplied any function, the **sum** function will be used



```
> operacion(1:10, mean)
[1] 5.5
> operacion(1:10, sum)
[1] 55
> operacion(1:10)
[1] 55
>
```

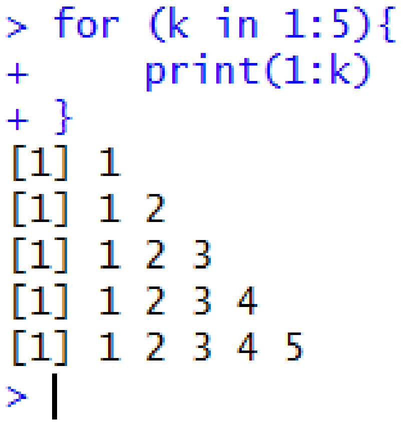
Exercise 23 (Solution)



```
> operacion <- function(input, f = sum) {
+     if (!is.numeric(input)) {
+         stop("El parámetro debe de ser númerico!")
+     }
+     f(input)
+ }
> operacion(1:10, mean)
[1] 5.5
> operacion(1:10, sum)
[1] 55
> operacion(1:10)
[1] 55
>
```

Loop for

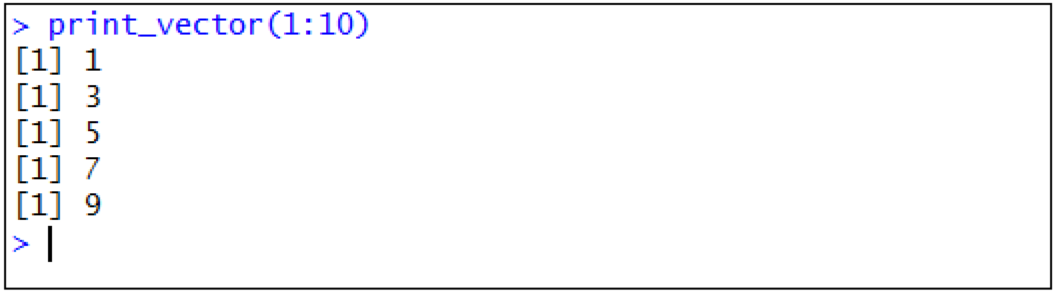
- Loop **for**() traverses a vector executing the commands found between the braces:



```
> for (k in 1:5){
+     print(1:k)
+ }
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
> |
```

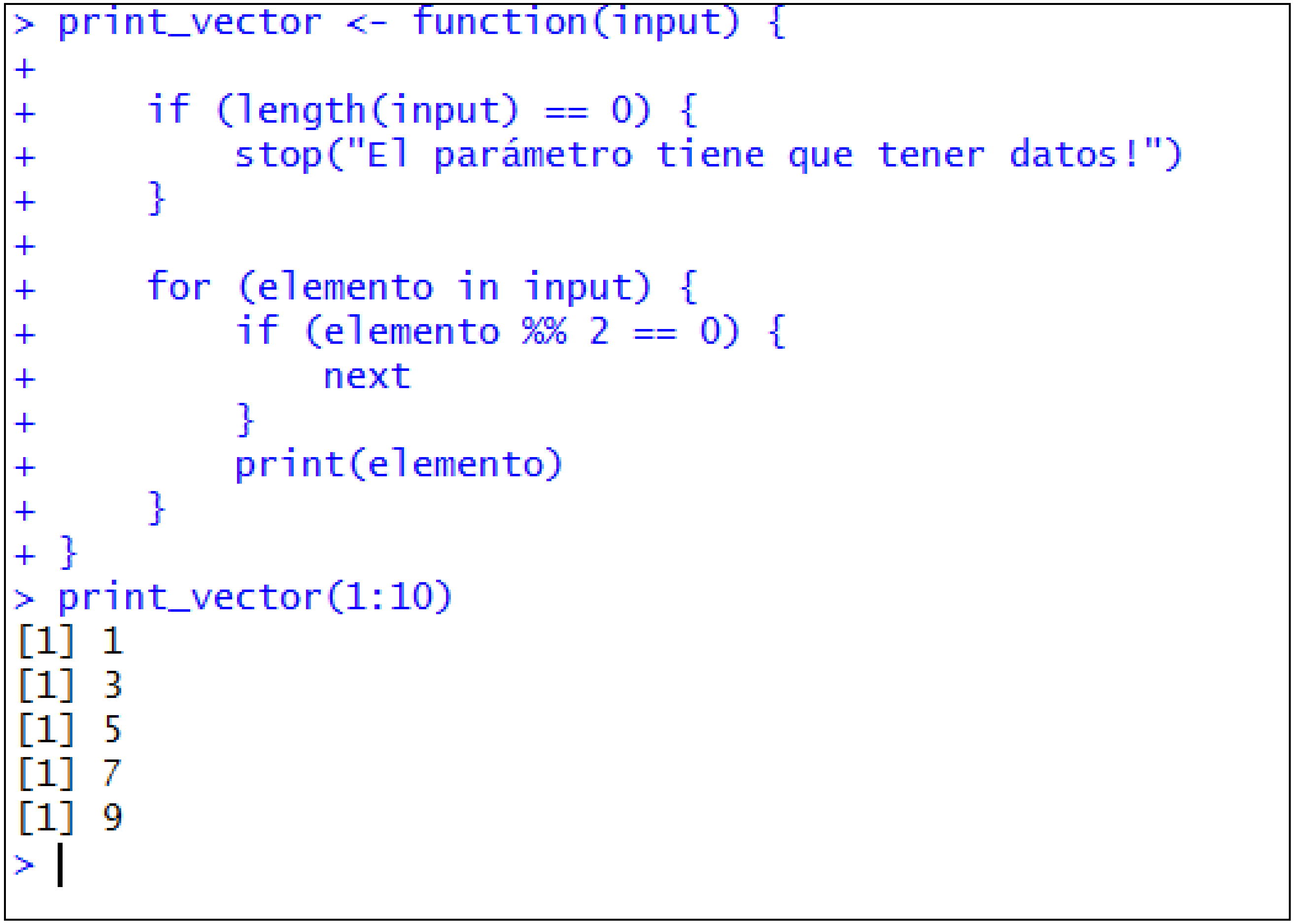
Exercise 24

- Create a function called "**print_vector**"Print on screen all the odd elements of vector passed as an argument



```
> print_vector(1:10)
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
> 
```

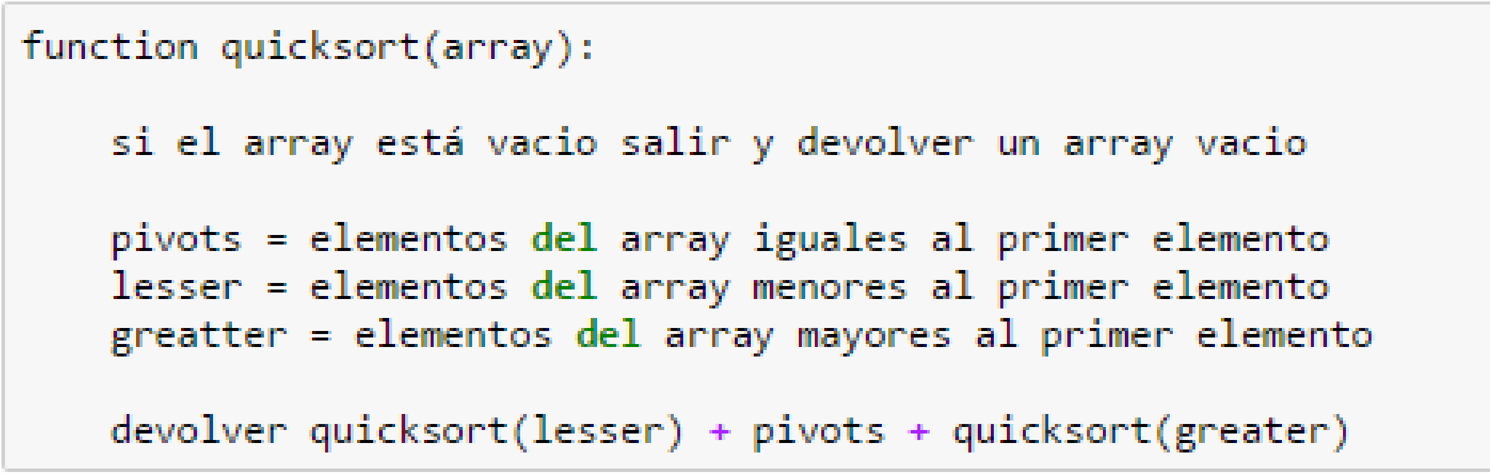
Exercise 24 (Solution)



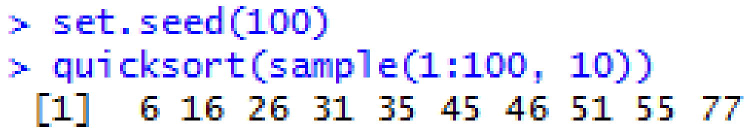
```
> print_vector <- function(input) {
+ 
+     if (length(input) == 0) {
+         stop("El parámetro tiene que tener datos!")
+     }
+ 
+     for (elemento in input) {
+         if (elemento %% 2 == 0) {
+             next
+         }
+         print(elemento)
+     }
+ }
> print_vector(1:10)
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
> 
```

Exercise 25

- Write a function in R implementing the QuickSort algorithm



```
function quicksort(array):

    si el array está vacio salir y devolver un array vacio

    pivots = elementos del array iguales al primer elemento
    lesser = elementos del array menores al primer elemento
    greatter = elementos del array mayores al primer elemento

    devolver quicksort(lesser) + pivots + quicksort(greater)
```



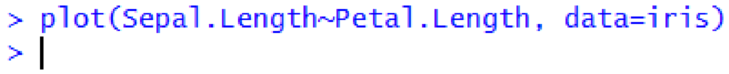
```
> set.seed(100)
> quicksort(sample(1:100, 10))
 [1]  6 16 26 31 35 45 46 51 55 77
```

Index

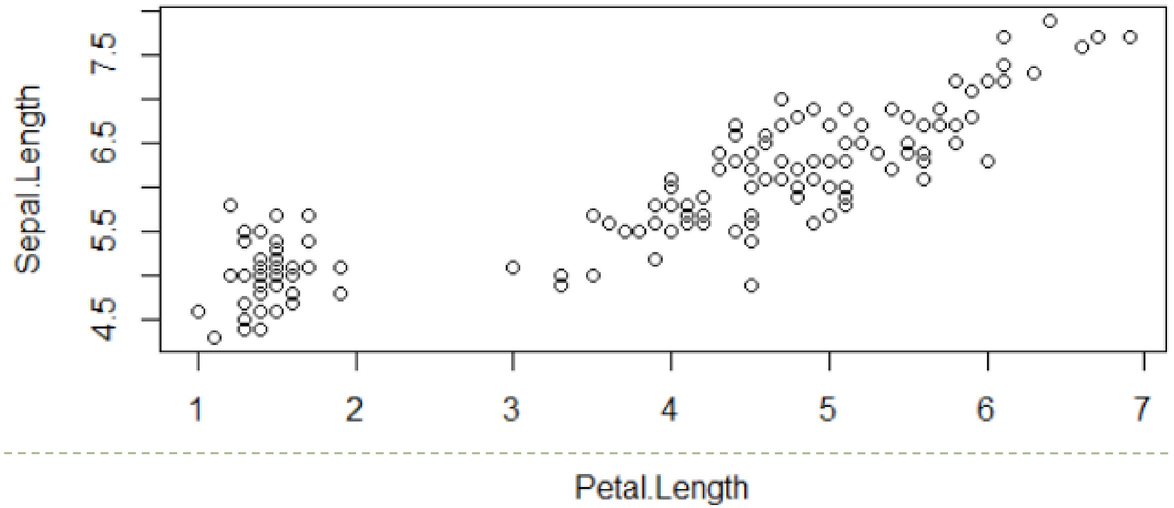
- Introduction
- Installing R
- R as a calculator
- Structures of data
- Statistical functions
- Probability distributions
- Writing functions
- Graphics in R

Graphics in R - The Base System

- It is the original R system and no additional packages need to be installed
- The idea is that you start with an empty canvas and from there graphic elements are added
- It is the most convenient for exploratory analysis of information

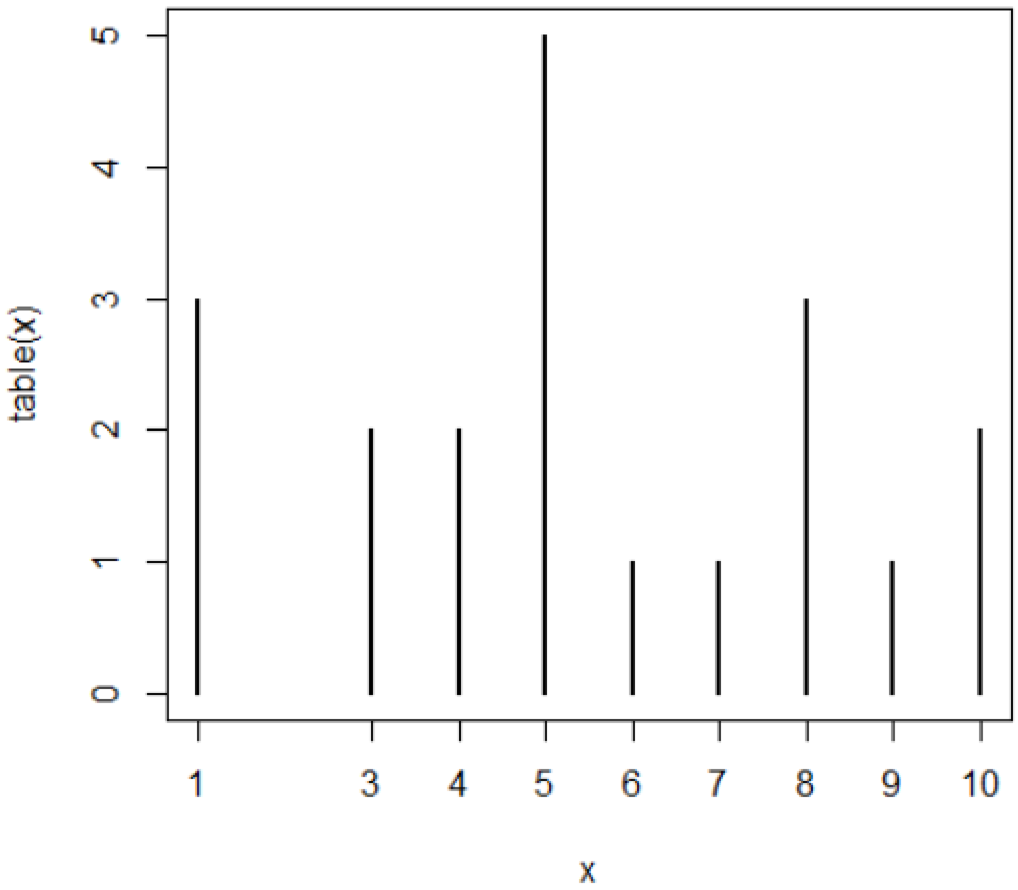


```
> plot(Sepal.Length~Petal.Length, data=iris)
> |
```

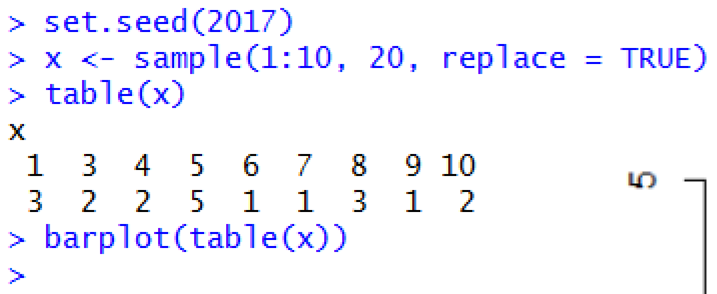


Bars

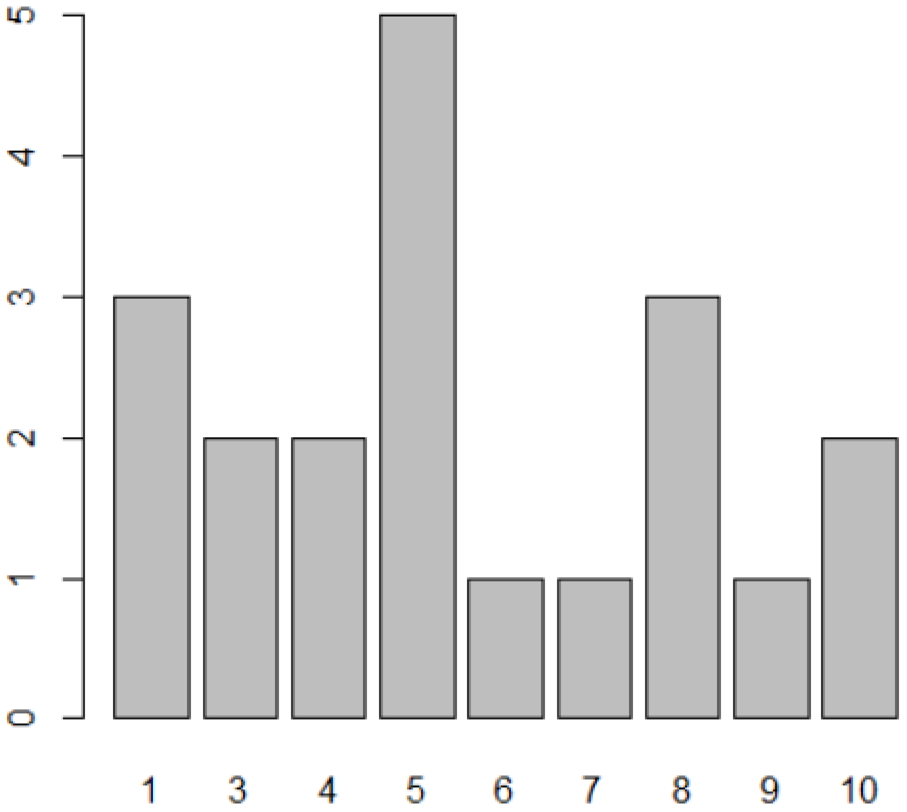
```
> set.seed(2017)
> x <- sample(1:10, 20, replace = TRUE)
> x
 [1] 10  6  5  3  8  8  1  5  5  3  7  1  1  5  5  4  4  8 10  9
> table(x)
x
 1  3  4  5  6  7  8  9 10 
 3  2  2  5  1  1  3  1  2 
> plot(table(x))
> 
```



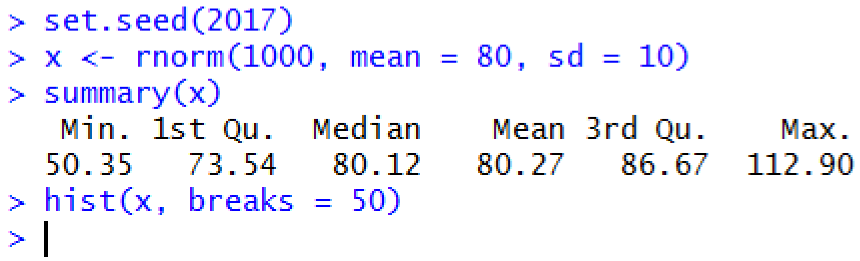
Bars



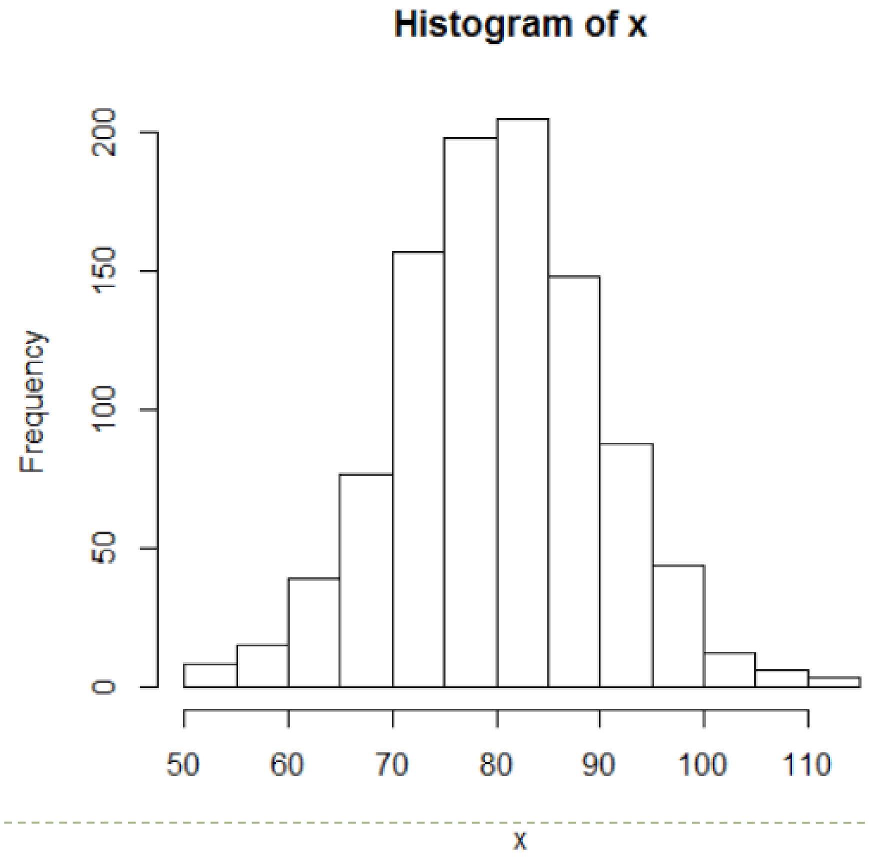
```
> set.seed(2017)
> x <- sample(1:10, 20, replace = TRUE)
> table(x)
x
 1  3  4  5  6  7  8  9 10 
 3  2  2  5  1  1  3  1  2 
> barplot(table(x))
> 
```



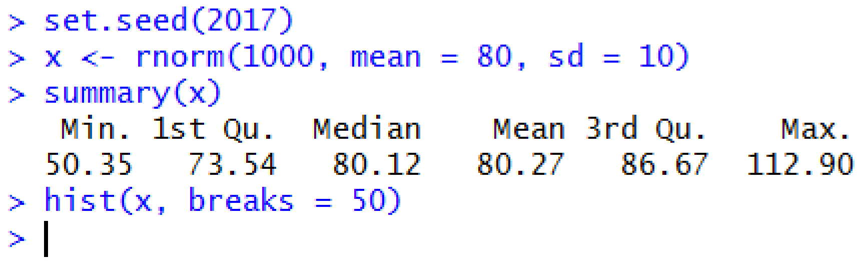
Histogram



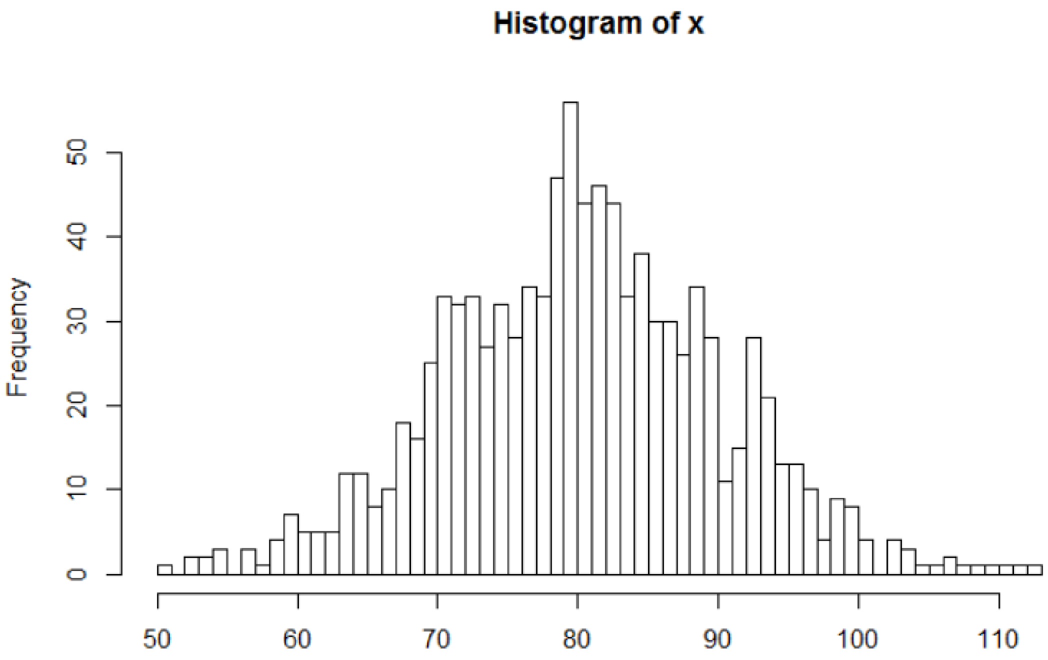
```
> set.seed(2017)
> x <- rnorm(1000, mean = 80, sd = 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.35   73.54   80.12   80.27   86.67  112.90 
> hist(x, breaks = 50)
> 
```



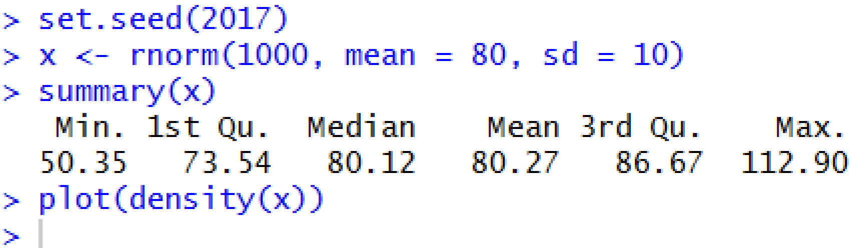
Histogram



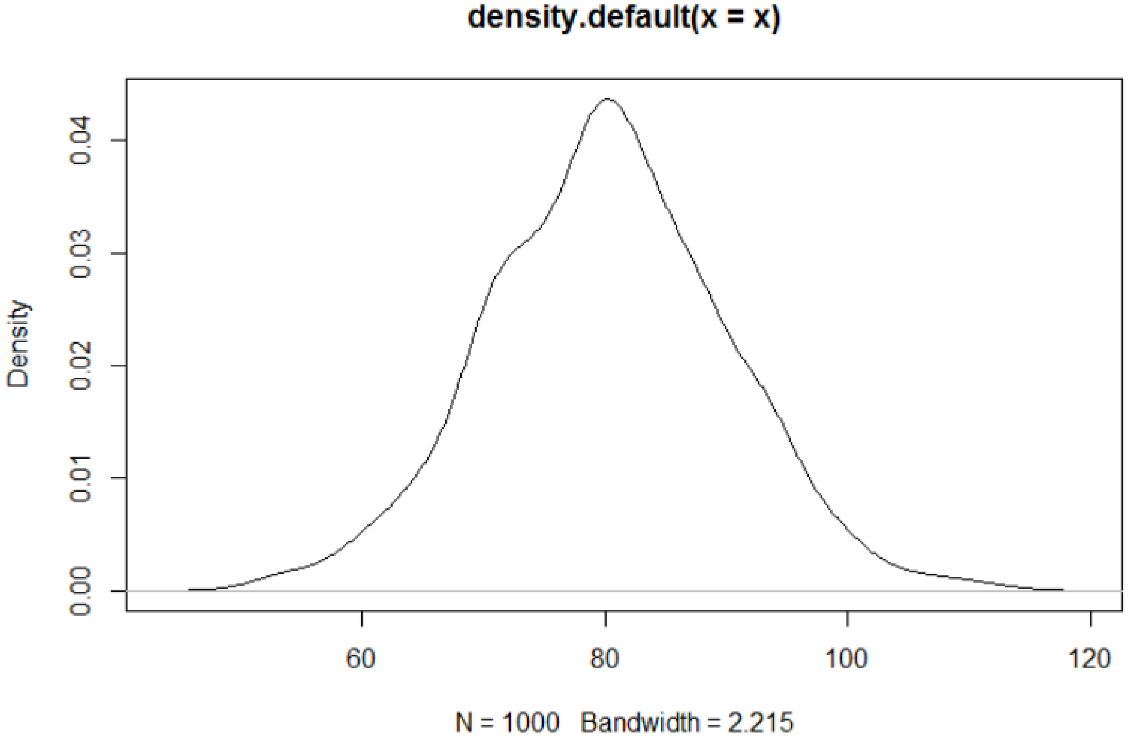
```
> set.seed(2017)
> x <- rnorm(1000, mean = 80, sd = 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.35   73.54   80.12   80.27   86.67  112.90 
> hist(x, breaks = 50)
> |
```



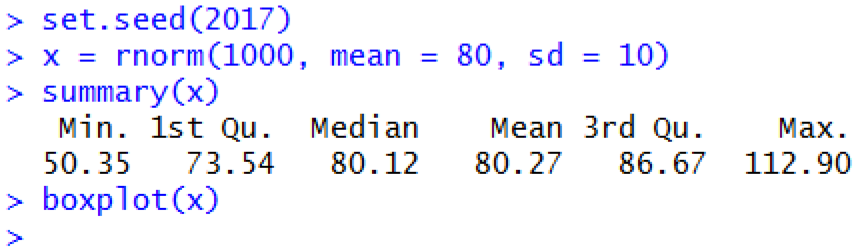
Density graph



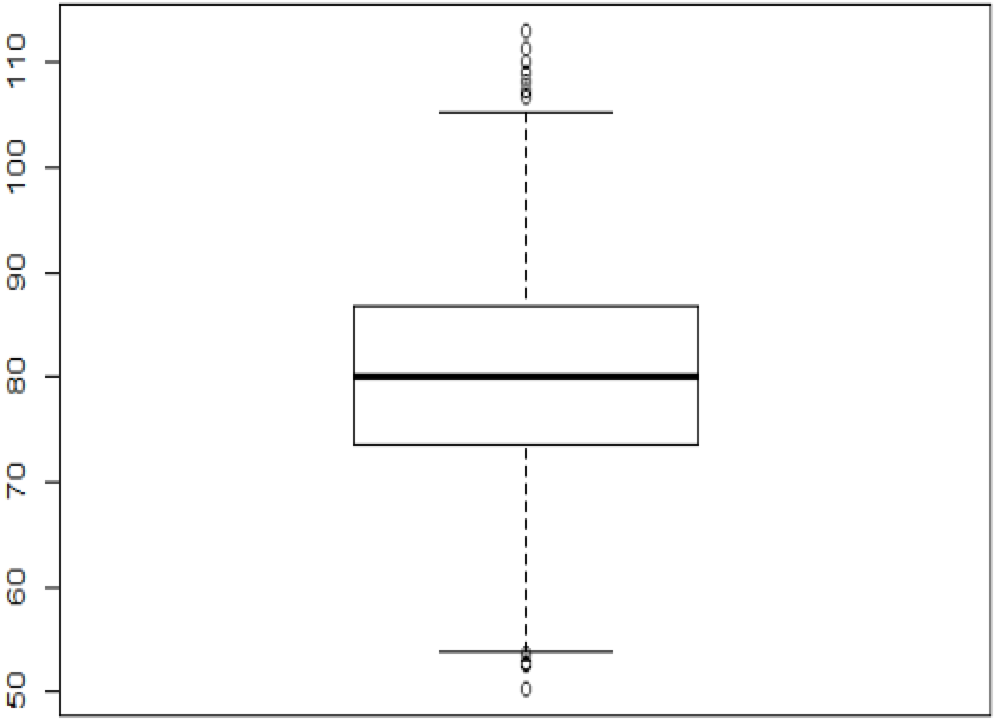
```
> set.seed(2017)
> x <- rnorm(1000, mean = 80, sd = 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.35   73.54   80.12   80.27   86.67  112.90 
> plot(density(x))
> 
```



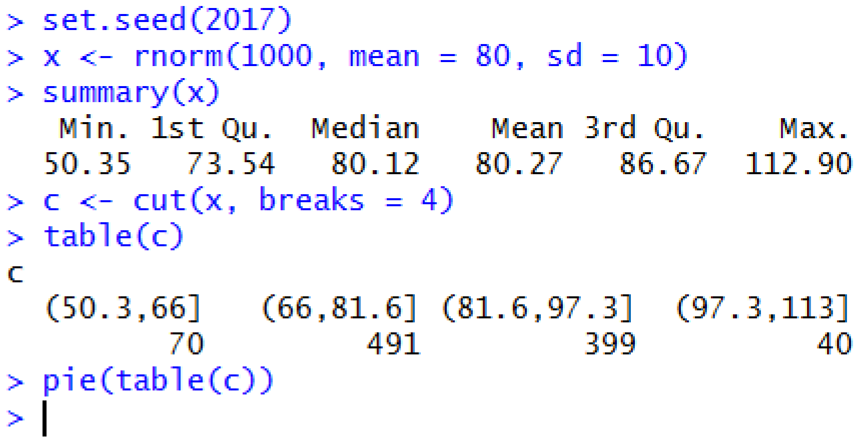
Box Plot



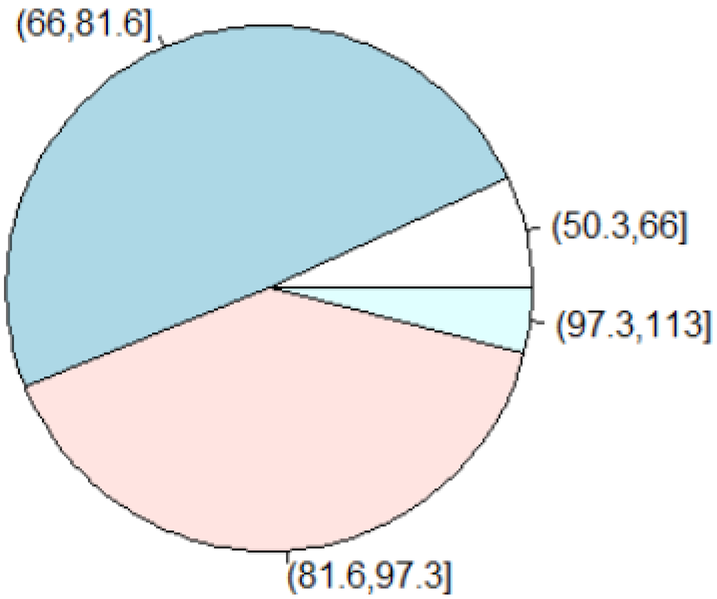
```
> set.seed(2017)
> x = rnorm(1000, mean = 80, sd = 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.35   73.54   80.12   80.27   86.67  112.90 
> boxplot(x)
> 
```



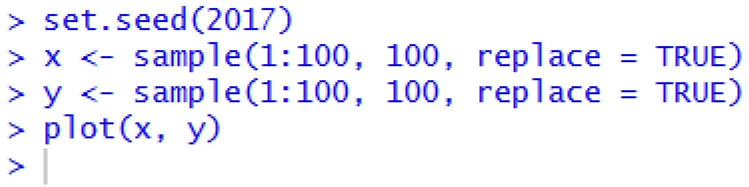
Pie Grapth



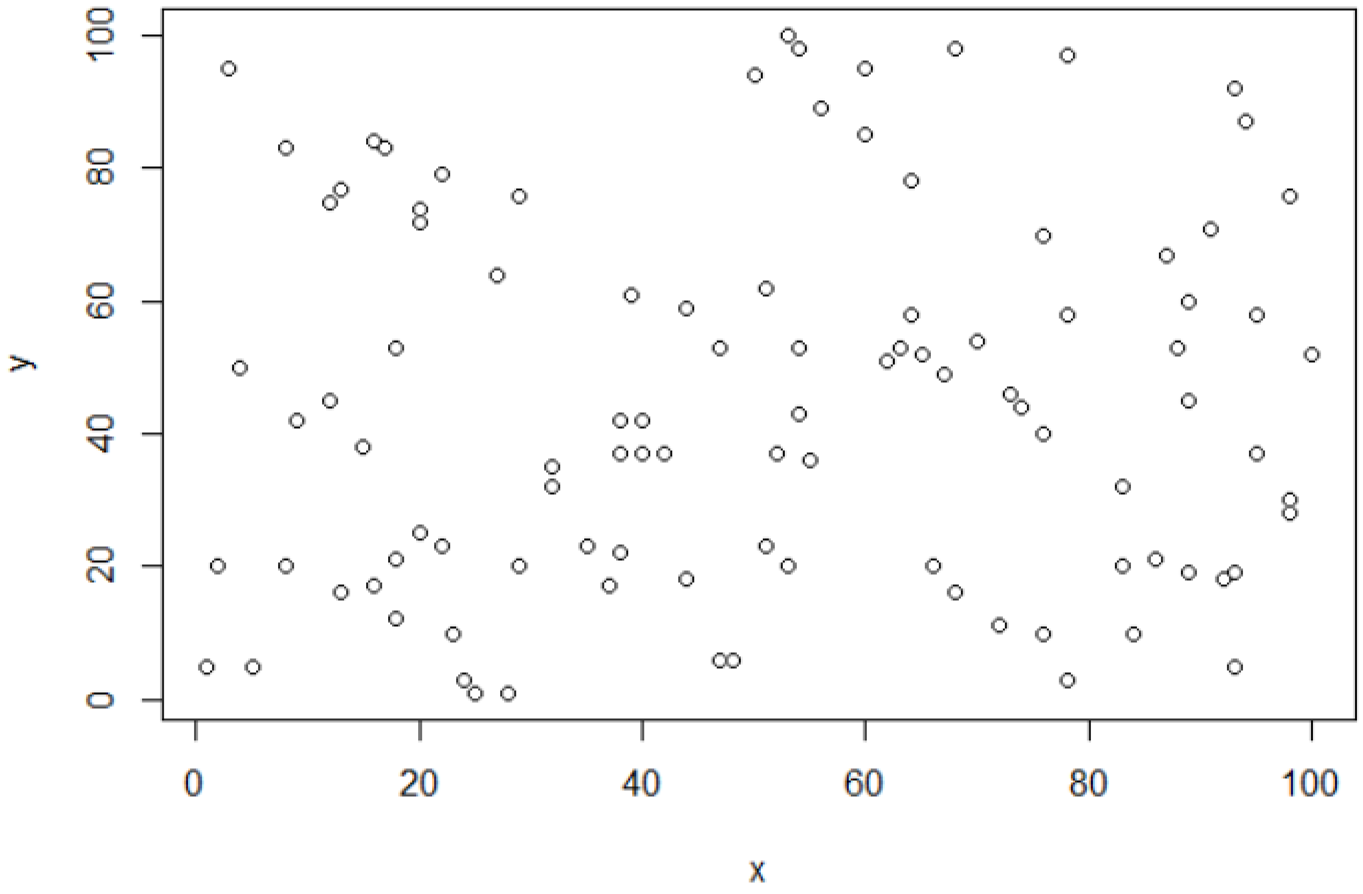
```
> set.seed(2017)
> x <- rnorm(1000, mean = 80, sd = 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.35   73.54   80.12   80.27   86.67  112.90 
> c <- cut(x, breaks = 4)
> table(c)
c
  (50.3,66]   (66,81.6] (81.6,97.3]  (97.3,113] 
         70         491         399          40 
> pie(table(c))
> 
```



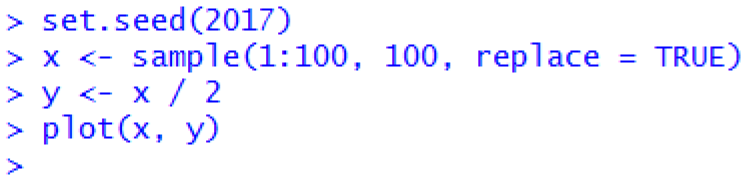
Scatter Plot



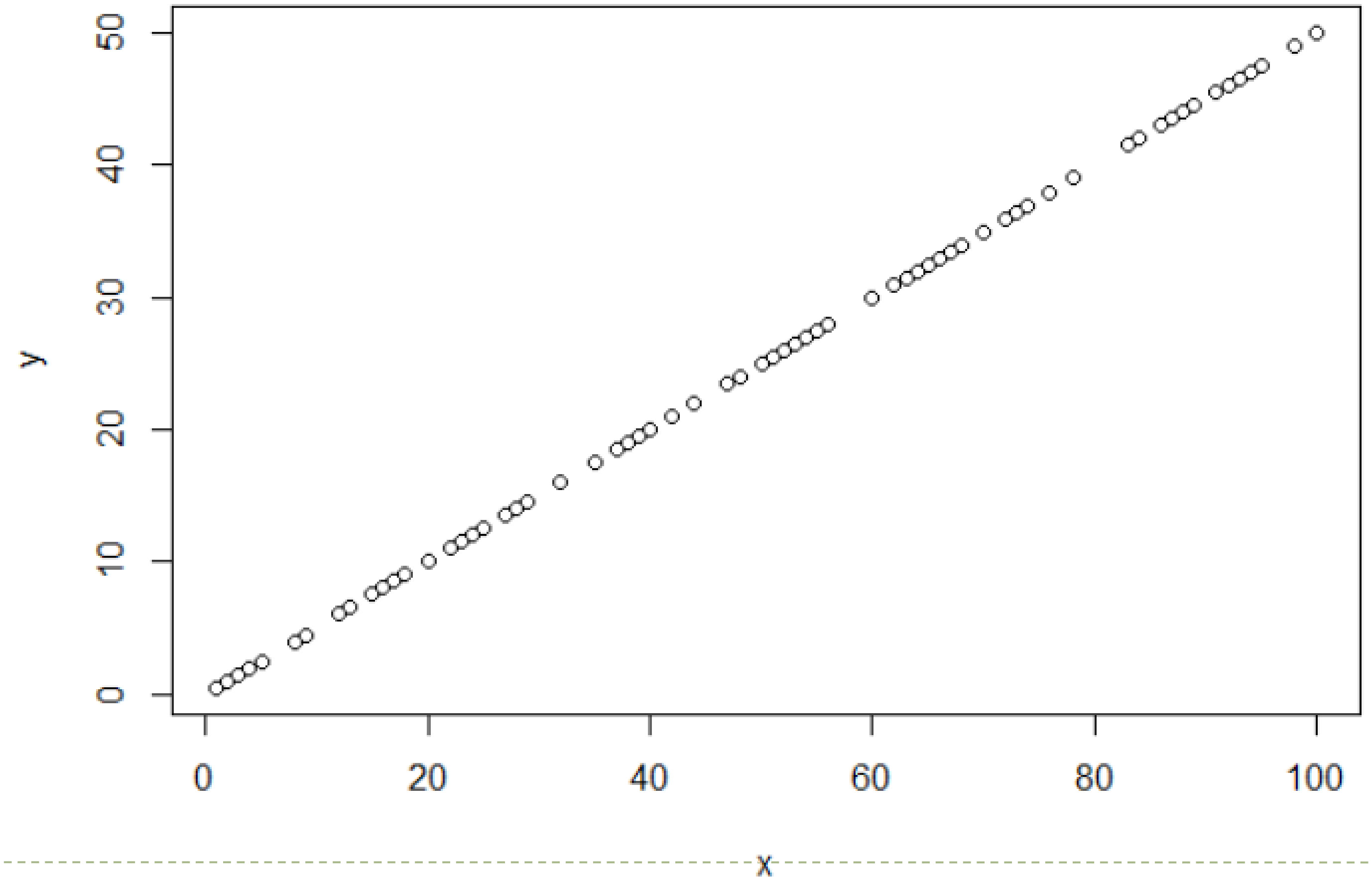
```
> set.seed(2017)
> x <- sample(1:100, 100, replace = TRUE)
> y <- sample(1:100, 100, replace = TRUE)
> plot(x, y)
> 
```



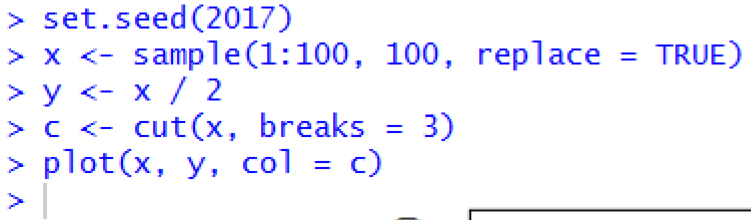
Scatter Plot



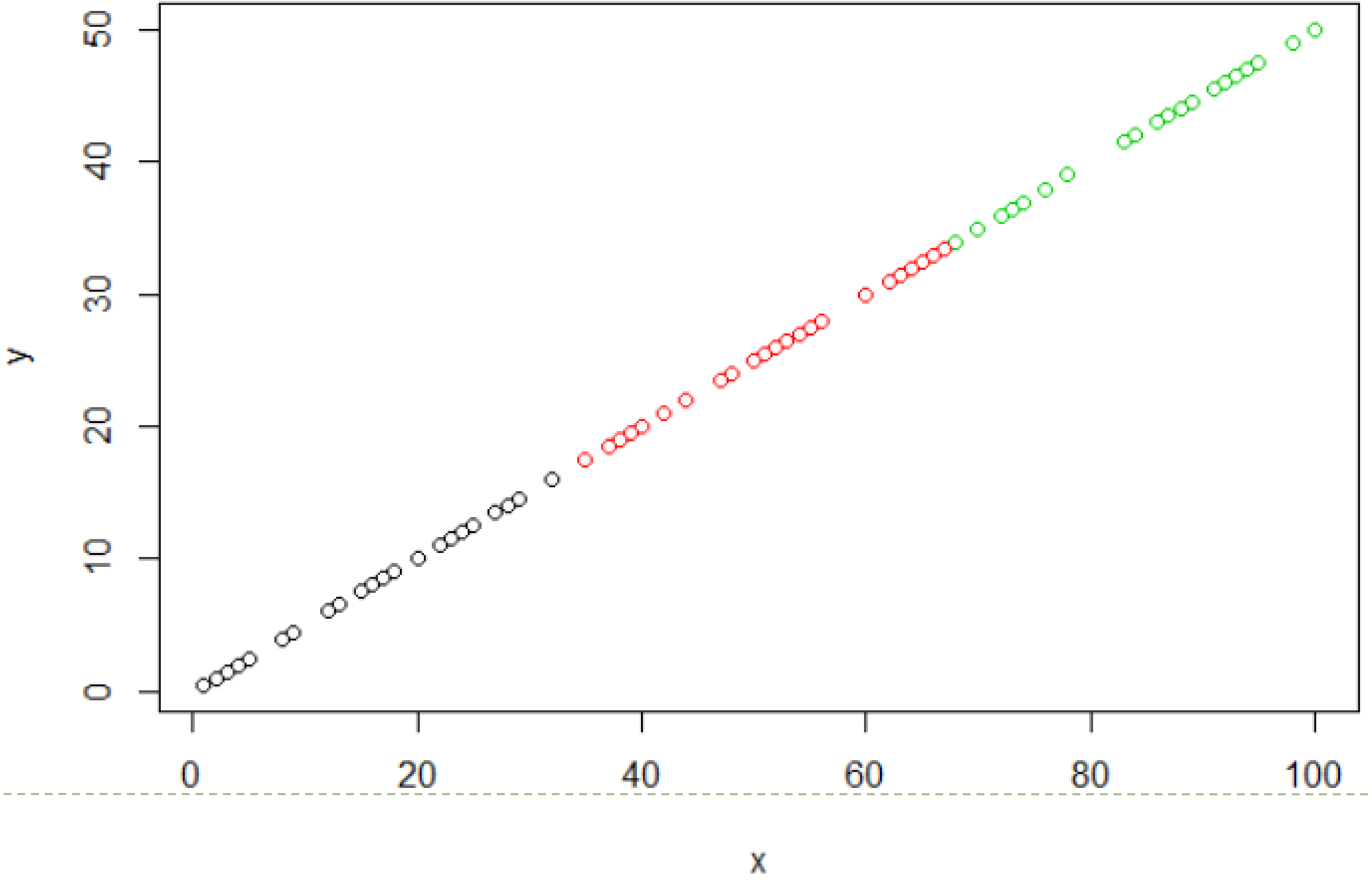
```
> set.seed(2017)
> x <- sample(1:100, 100, replace = TRUE)
> y <- x / 2
> plot(x, y)
>
```



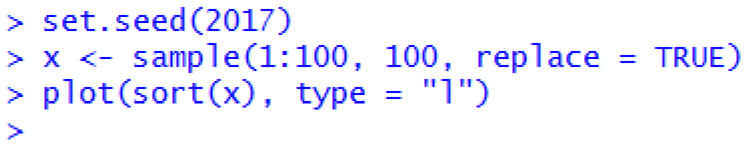
Scatter Plot



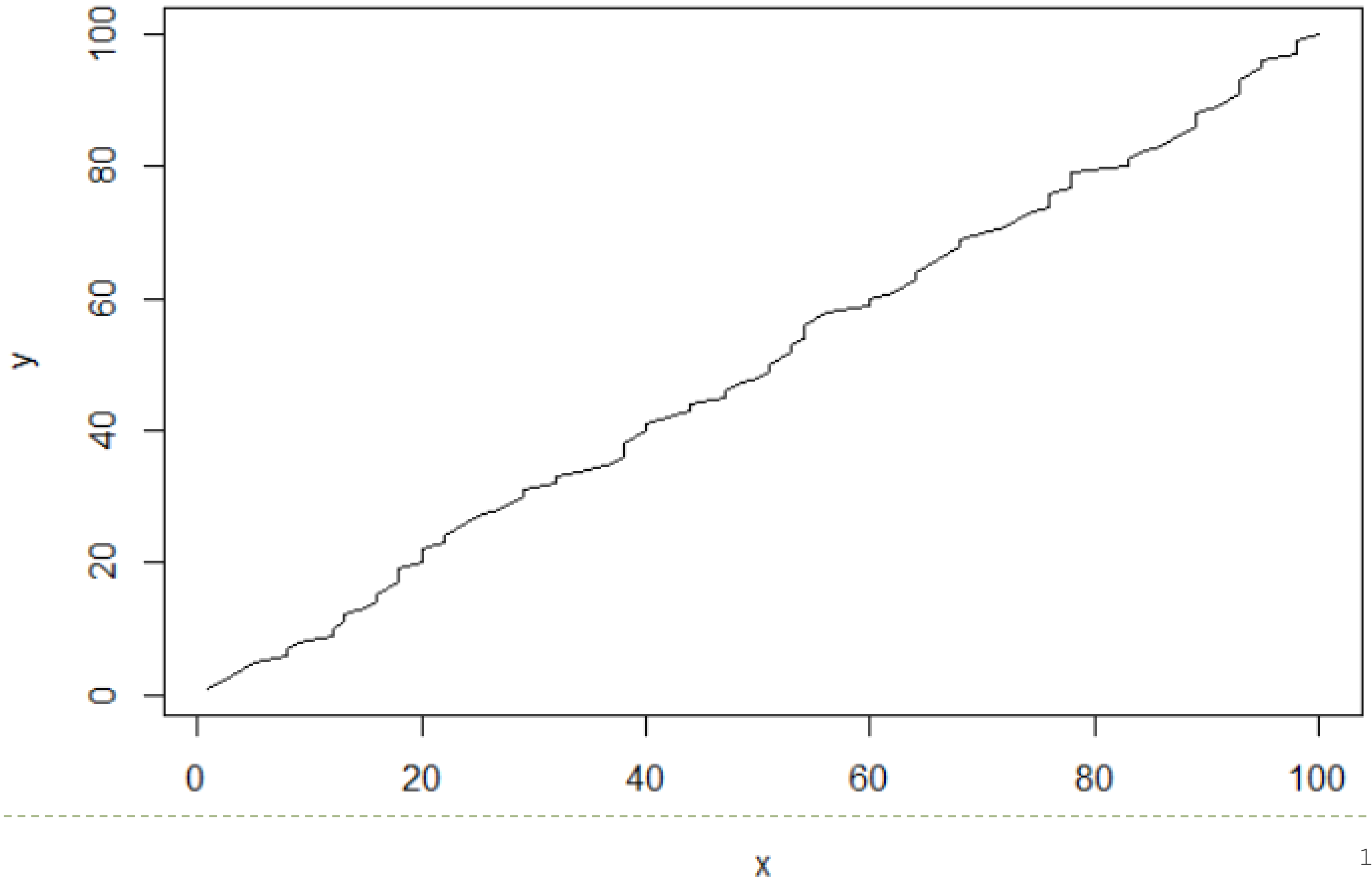
```
> set.seed(2017)
> x <- sample(1:100, 100, replace = TRUE)
> y <- x / 2
> c <- cut(x, breaks = 3)
> plot(x, y, col = c)
> 
```



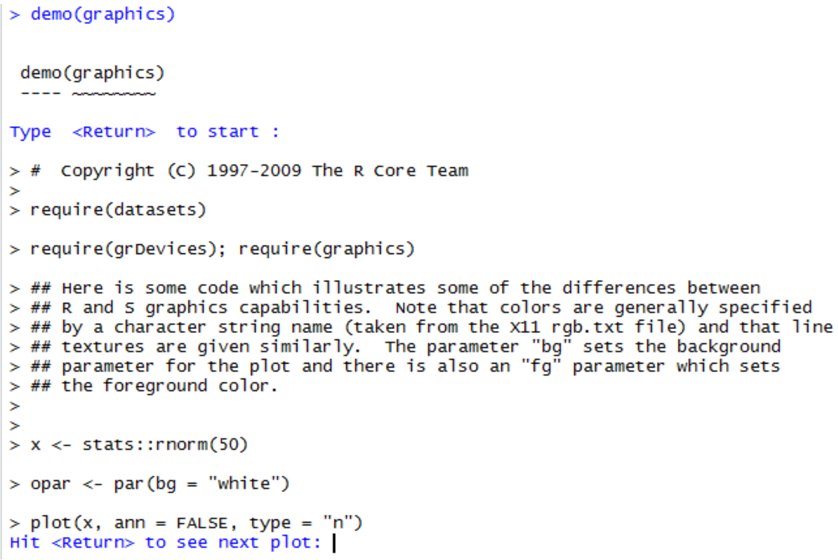
Line Grapth



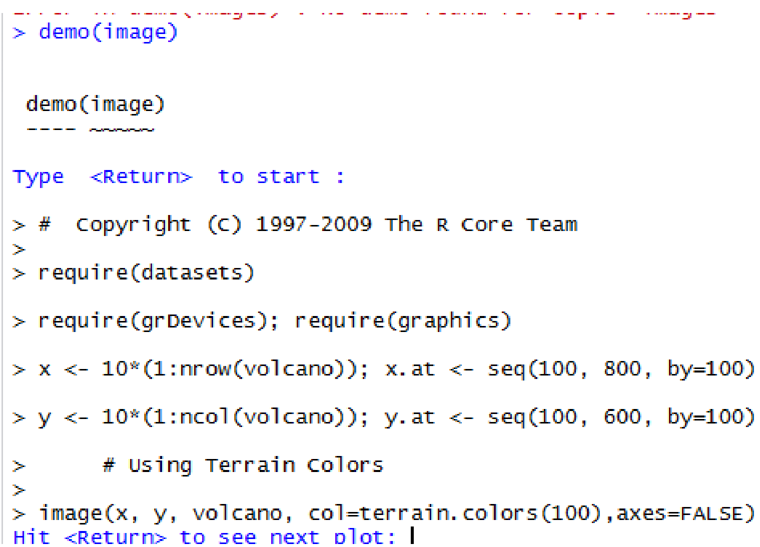
```
> set.seed(2017)
> x <- sample(1:100, 100, replace = TRUE)
> plot(sort(x), type = "l")
>
```



Demos



```
> demo(graphics)


 demo(graphics)
 ---- ~~~~~~~~

Type  <Return>  to start :

> #  Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

> ## Here is some code which illustrates some of the differences between
> ## R and S graphics capabilities.  Note that colors are generally specified
> ## by a character string name (taken from the X11 rgb.txt file) and that line
> ## textures are given similarly.  The parameter "bg" sets the background
> ## parameter for the plot and there is also an "fg" parameter which sets
> ## the foreground color.
>
>
> x <- stats::rnorm(50)

> opar <- par(bg = "white")

> plot(x, ann = FALSE, type = "n")
Hit <Return> to see next plot:
```



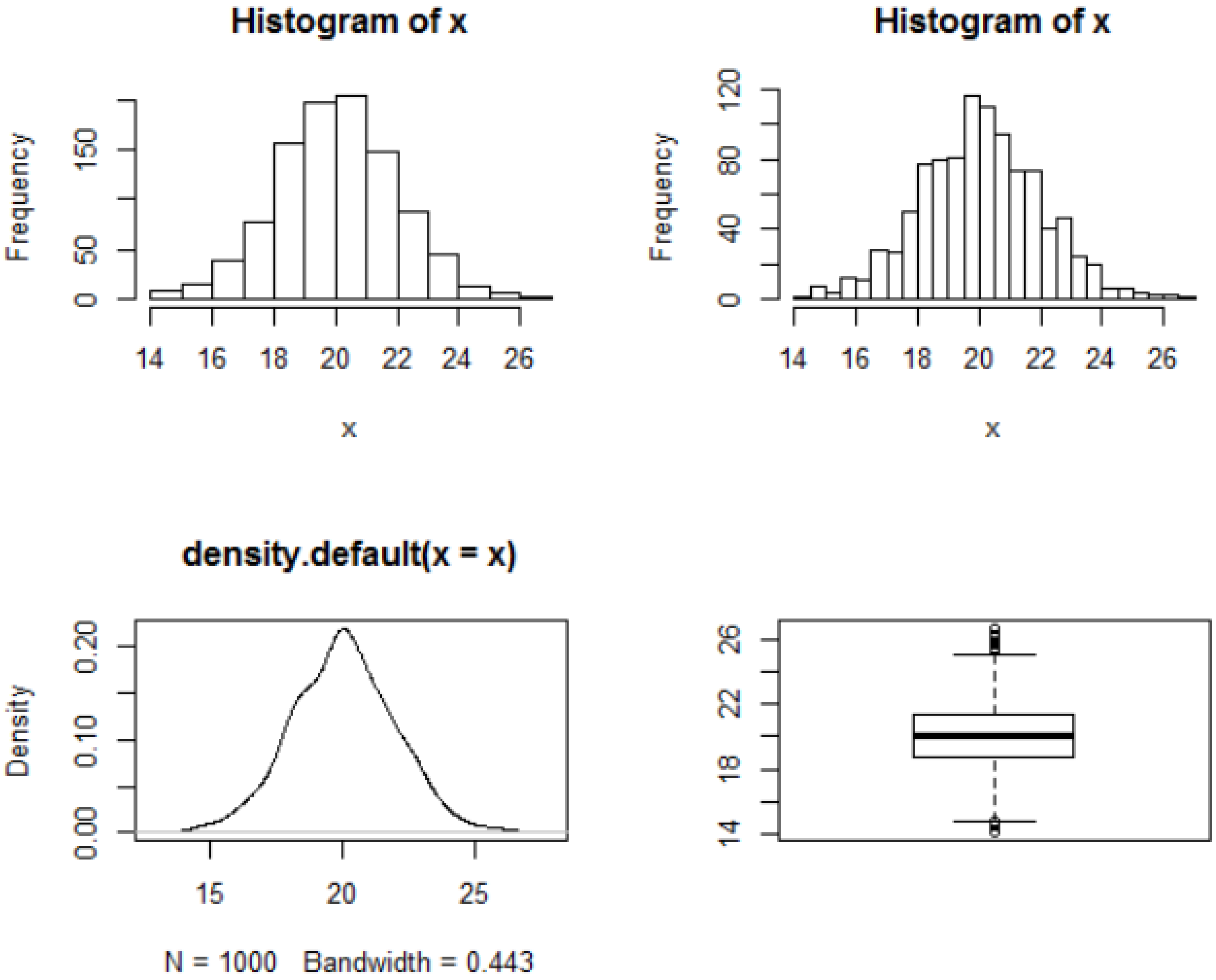
```
> demo(image)


 demo(image)
 ---- ~~~~~

Type  <Return>  to start :

> #  Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

> x <- 10*(1:nrow(volcano)); x.at <- seq(100, 800, by=100)

> y <- 10*(1:ncol(volcano)); y.at <- seq(100, 600, by=100)

>      # Using Terrain Colors
>
> image(x, y, volcano, col=terrain.colors(100),axes=FALSE)
Hit <Return> to see next plot:
```

Exercise 26

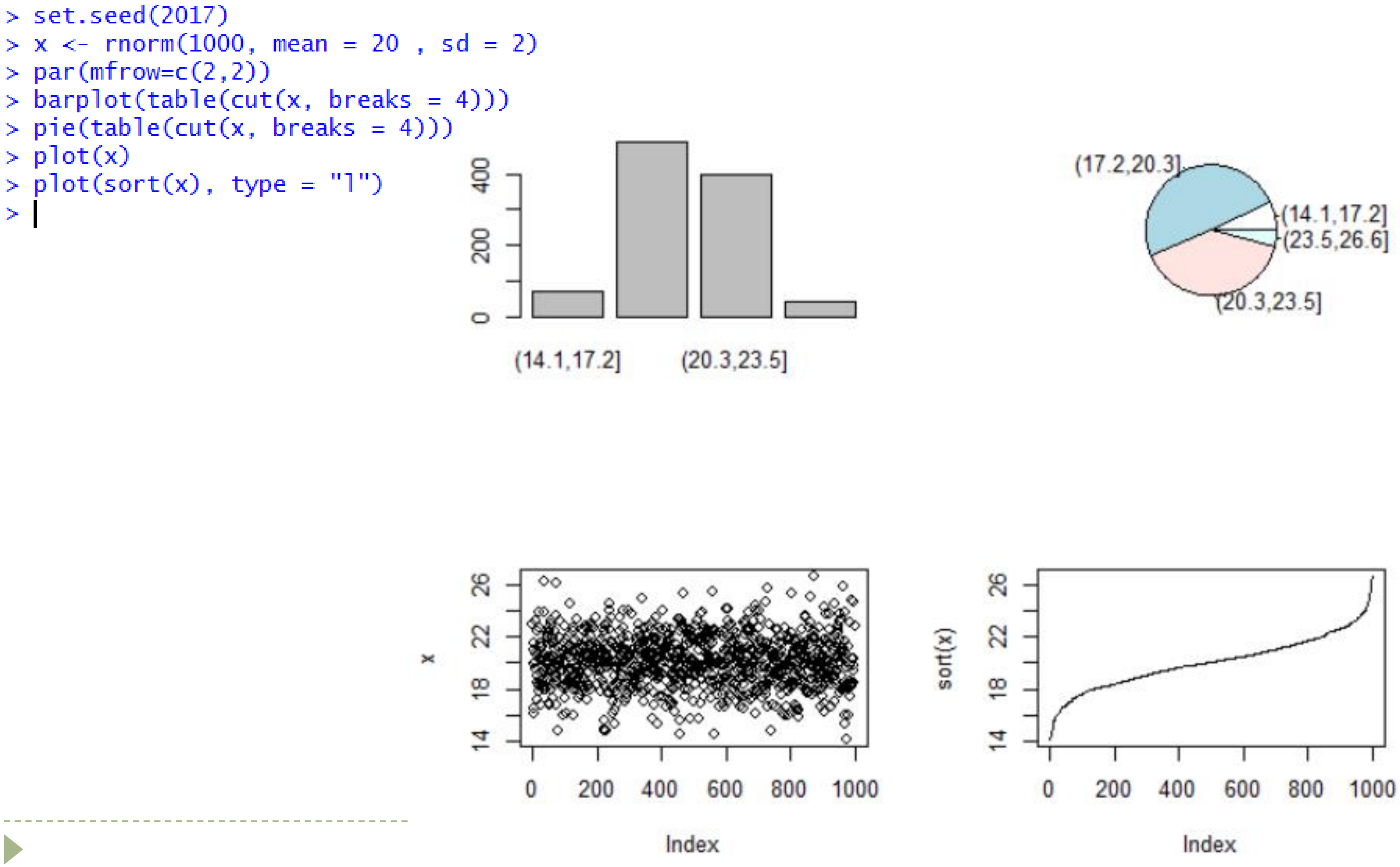
- Set the random seed to 2017
- Create a vector of numbers according to the normal distribution (mean = 20 and sd = 2)
- With this data create 8 different graphs
    - 2 different types of histograms
    - 1 density graph
    - 1 box graph
    - 1 pie chart
    - 1 bar graph
    - 1 scatter plot
    - 1 line graph

Exercise 26 (Solution)

```
> set.seed(2017)
> x <- rnorm(1000, mean = 20 , sd = 2)
> par(mfrow=c(2,2))
> hist(x)
> hist(x, breaks = 20)
> plot(density(x))
> boxplot(x)
>
```



Exercise 26 (Solution)



```
> set.seed(2017)
> x <- rnorm(1000, mean = 20 , sd = 2)
> par(mfrow=c(2,2))
> barplot(table(cut(x, breaks = 4)))
> pie(table(cut(x, breaks = 4)))
> plot(x)
> plot(sort(x), type = "l")
> |
```

THANKS FOR YOUR ATTENTION

Daniel Villanueva Jiménez
daniel.villanueva@immune.institute

@dvillaj