Lists

- Lists can be used to combine objects (of possibly different kinds/sizes) into a larger composite object.
- The components of the list are named according to the arguments used.
- Components can be extracted with the double bracket operator `[[ ]]`
- Alternatively, named components can be accessed with the "$" separator.

```
> A<-c(31,32,40)
> S<-as.factor(c("F","M","M","F"))
> People<-list(age=A,sex=S)
> People
$age
[1] 31 32 40
$sex
[1] F M M F
Levels: F M
```

```
Indexing Lists

> People[[1]]
[1] 31 32 40

> People$age
[1] 31 32 40
```

Names

Names of an R object can be accessed and/or modified with the `names()` function.

```
z <- list(a = 1, b = "c", c = 1:3)
> z
$a
[1] 1

$b
[1] "c"

$c
[1] 1 2 3
```

```
# change just the name of the third element.
names(z)[3] <- "c2"
z

$a
[1] 1

$b
[1] "c"

$c2
[1] 1 2 3
```

Input/Output: Keyboard and Monitor

- Suppose we have a file (file.txt) with this content:

```
12
2 5
641
```

```
scan("file.txt")
Read 4 items
[1]  12   2   5 641
scan("file.txt",what=character())
Read 4 items
[1] "12"  "2"   "5"   "641"
scan("file.txt",sep="\n")
Read 3 items
[1]  12  25 641
```

Input/Output: Keyboard and Monitor

- Use `scan()` to read from the keyboard by specifying an empty string for the filename:

```
scan("")
1: 23 4
3: 2
4:
Read 3 items
[1] 23  4  2
```

- Note that we are prompted with the index of the next item to be input, and we signal the end of input with an empty line.

Input/Output: Keyboard and Monitor

- To read in a single line from the keyboard use `readline()`:

```
readline("Input data: ")
Input data: 23 4 2
[1] "23  4  2"
```

- Note that we are prompted with the index of the next item to be input, and we signal the end of input with an empty line.

Input/Output: Print to the screen

- `print()` is a *generic* function, so the function call depends on the class of the object that is printed.
- If, for example, the argument is of class table, then the `print.table()` function will be called.

```
x <- 1:3

print(x^2)

[1] 1 4 9
```

Input/Output: Print to the screen

- It is better to use `cat()` instead of `print()`, as the latter can print only one expression and its output is numbered:

```
x <- 1:3
print(x^2)
[1] 1 4 9
cat(x^2)
1 4 9
cat(x^2, x, "hola")
1 4 9 1 2 3 hola
cat(x^2, x, "hola", sep="_")
1_4_9_1_2_3_hola
```

Input/Output: Reading and Writing files

- We will use of the function `read.table()` to read in a data frame.
- Suppose we have a file `matrix.txt` with the following content:

```
nombre edad
John 25
Mary 28
Jim 19
```

Input/Output: Reading and Writing files

- The first line contains an optional header, specifying column names. We could read the file this way:

```
read.table("matrix.txt",header=TRUE)
```

```
  nombre edad
1   John   25
2   Mary   28
3    Jim   19
```

- Note that `scan()` would not work here, because our file has a mixture of numeric and character data (and a header).

Input/Output: Reading and Writing files

- If we want to write a file, we change `read.table()` for `write.table()` function:

```
write.table(matrix(1:6, nrow=2), "output.txt",
row.names=FALSE, col.names=FALSE)
```

output.txt:

```
1 3 5
2 4 6
```

Input/Output: Reading and Writing files

- The function `cat()` can also be used to write to a file, one part at a time:

```
cat("abc\n",file="u.txt")
cat("de\n",file="u.txt",append=TRUE)
```

```
u.txt:
abc
de
```

Exporting Data: cat()

R objects can be exported to a text file using the cat() function:

```
cat (x , file = "", sep = " ", fill = FALSE, labels
= NULL, append = FALSE)
```

x: R object

file: character string naming the file to print to. If "" (the default), cat prints to the console unless redirected by sink.

sep: a character vector of strings to append after each element

fill: controls how the output is broken into successive lines.

append: logical. If TRUE output will be appended to file; otherwise, it will overwrite the contents of file.

Importing data

Read data from an excel: `read.csv()`

- Use R to read the file in .csv format:

```
# first row contains variable names, comma is
separator
# assign the variable id to row names
# note the "/" instead of "\" on mswindows systems

mydata <- read.csv("c:/mydata.csv", header=TRUE,
sep=",", row.names="id")
```

read.delim()

- They are intended to read TAB separated files

```
read.delim(file, header = TRUE, sep = "\t", dec=".",
fill =TRUE, ...)
```

- sep: the field separator character. “\t” (default for read.delim) stands for TAB separator;

- fill: if TRUE then in case the rows have unequal length, blank fields are implicitly added

Read data from an excel: `read.csv2()`

- Use R to read the file in .csv format from countries that use a comma (“,”) as decimal point and a semicolon (“;”) as field separator.

```
# first row contains variable names, comma is
separator
# assign the variable id to row names
# note the “/” instead of “\” on mswindows systems

mydata <- read.csv2("c:/mydata.csv", header=TRUE,
sep=“;", dec=",“, row.names="id")
```

Read data from SPSS: `spss.get()`

```
# Import international.sav as a data frame: demo
demo <- read.spss("international.sav",
to.data.frame = TRUE)
```

Read data from SPSS: `read.dta()`

```
# input Stata file
library(foreign)
mydata <- read.dta("c:/mydata.dta")
```

Read data from JSON Files: `fromJSON()`

```
# Activate `rjson`
library(rjson)

# Import data from json file
JsonData <- fromJSON(file= "<fichero.json>" )

# Import data from json file through an URL
JsonData
<- fromJSON(file= "<URL al fichero JSON >" )
```

Exporting Data

There are numerous methods for exporting **R** objects into other formats . For SPSS, SAS and Stata you will need to load the **foreign** packages. For Excel, you will need the **xlsReadWrite** package.

- **To an Excel Spreadsheet**

```
library(xlsReadWrite)write.xls(mydata, "c:mydata.xls")
```

Writing data frames

- write() writes out a matrix or vector in a specified number of columns.

- write.table() writes out a data frame (or an object that can be coerced to a data frame) with row and column labels

```
write.table(mydata, "c:/mydata.txt", sep="\t")
```

```
write.table(x, file = "", append = FALSE, sep = "
“, na = "NA", dec = ".", row.names = TRUE,
col.names = TRUE)
```

Source Codes: Input

The input can come from a script file (a file containing **R** commands)

The **source( )** function runs a script in the current session. If the filename does not include a path, the file is taken from the current working directory.

```
# input a script
source("myfile")
```

Source Codes: Output

The **sink( )** function defines the direction of the output.

```
# output directed to output.txt
# output overwrites existing file. no output to
   terminal.
   sink("myfile.txt", append=TRUE, split=TRUE)

# return output to the terminal
   sink()
```

General Subsetting Rules

Subsetting syntax:

```
# Subsetting of one dimensional objects (e.g.vectors,
factors)

my_object[row]

# Subsetting of two dimensional objects,(e.g. matrices,data
frames).

my_object[row, col]

# Subsetting of three dimensional objects, like arrays.
my_object[row, col, dim]
```

General Subsetting Rules

There are three possibilities to subset data objects

(1) Subsetting by positive or negative index/position numbers

```
# Creates a vector sample with named elements.
my_object <- 1:26;
names(my_object) <- LETTERS

# Returns the elements 1-4.
my_object[1:4]

# Excludes elements 1-4.
my_object[-c(1:4)]
```

General Subsetting Rules

There are three possibilities to subset data objects

(2) Subsetting by same length logical vectors

```
# Generates a logical vector as example.
my_logical <- my_object > 10

# Returns the elements where my_logical contains TRUE
values.
my_object[my_logical]
```

(3) Subsetting by field names

```
# Returns the elements with element titles: B, K, M
my_object[c("B", "K", "M")]
```

Summary

- R is an interactive statistical language
- Extremely flexible and powerful
- Data manipulation and coding
- Can be used as a calculator, simulator and sampler
- FREE!

Gracias…