Basic R functionality

Data structures and flow control

Tom.Van.Engeland@gmail.com

Getting help

Function is known exactly (standard help function):

```
?plot

?rnorm

?<-
```

Function is not known more or less (fuzzy search):

```
??norm

??operators
```

No functions known:

Assignments

Standard assignments: `->`

* Anywhere applicable
* Leftward or rightward
* Example:

```
a <- 6
20 -> b
a <- b <- c <- 4
```

= operator:

* Only applicable at toplevel of environment or in list of expessions
* Example:

```
a = 6
a <- rnorm(n = 6,mean = 4,sd = 2)
```

Global assignments: `->>`

* Only used in function to assign value to non-local variable
* Left or rightward
* Example:

```
a <<- 6

Fun <- function () {
      a <- 0
      B <<- 5
}
```

Exploring working environment

List of variables in the environment:

```
# see ?ls for more info

ls()
```

Removing a variable from the environment:

```
# see ?rm for more info

rm(a)             # no quotes

# check result with ls
ls()

# remove all variables
rm(list=ls())
```

Data types and structures

Vectors

Types (no explicit declaration needed):

"integer"
```
a <- 1:10
```

"double"
```
a <- seq(from = 0, to = 1, by= 0.01)
```

"character"
```
a <-
c("Winter","Spring","Summer","Autumn")
```

"logical"
```
a <- rep(c(TRUE,FALSE,TRUE),times=3)
```

. . .

Vectors have a length:
```
a <- 1:10
a
[1]  1  2  3  4  5  6  7  8  9 10

length(a)
[1] 10
```

Elements are indexed:
```
a <-
c("Winter","Spring","Summer","Autumn")
a[2]
[1] "Spring"
a[c(2,4)]
[1] "Spring"  "Autumn"
```

Data types and structures

Vectors

Elements can be named...

```
a <- c(Winter = 1, Spring = 2, Summer = 3, Autumn =
4)
a
Winter Spring Summer Autumn
     1      2      3      4
a["Winter"]
Winter
     1
names(a)
[1] "Winter" "Spring" "Summer" "Autumn"

names(a) <-
c("season1","season2","season3","season4")
a
season1 season2 season3 season4
      1       2       3       4
names(a)
[1] "season1" "season2" "season3" "season4"
```

… and names can be used for indexing:

```
a["season1"]
season1
      1
```

Data types and structures

Matrices

a

1	4	7
2	5	8
3	6	9

construction:

```
a <- matrix(data= 1:9, ncol=3, nrow=3,byrow = FALSE,dimnames =
NULL)
a
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

matrix = 2-dimensional vector:

```
a <- rbind(1:5, 5:1)                    # explore cbind:   ?cbind
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    5    4    3    2    1

dim(a)
[1] 2 5

a <- 1:9
dim(a) <- c(3,3)
```

row and column names:

```
rownames(a) <- c("a","b","c") ; colnames(a) <- c("x","y","z")
dimnames(a) <- list(c("a","b","c"),c("x","y","z"))
```

Data types and structures

a

1	4	7
2	5	8
3	6	9

Matrices

Elementwise operations:

```
a <- matrix(data=1:9,ncol=3)
b <- matrix(data=1:9,ncol=3)

a + b     # element-wise addition
a - b     #       "      subtraction
a * b     #       "      multiplication
a / b     #       "      division
```

Matrix operations:

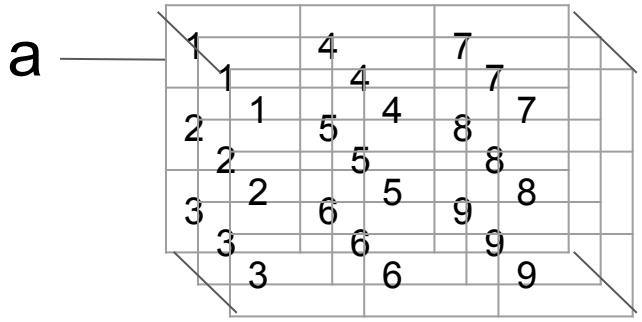
```
a <- matrix(data=1:10, ncol=5)
b <- t(a) # transpose
c <- matrix(c(1,4,2,7,2,9,10,20,42),ncol=3)

a %*% b    # matrix multiplication
det(a)     # determinant

solve(c)   # inversion
```

Indexing:

```
a[2,2]
[1] 5
A[“a”,”x”]
[1] 1
```

Data types and structures

Arrays



Generalisation to n dimensions:

```
a <- array(data=1:27, dim=c(3,3,3), dimnames=NULL)
```

Elementwise operations are still valid

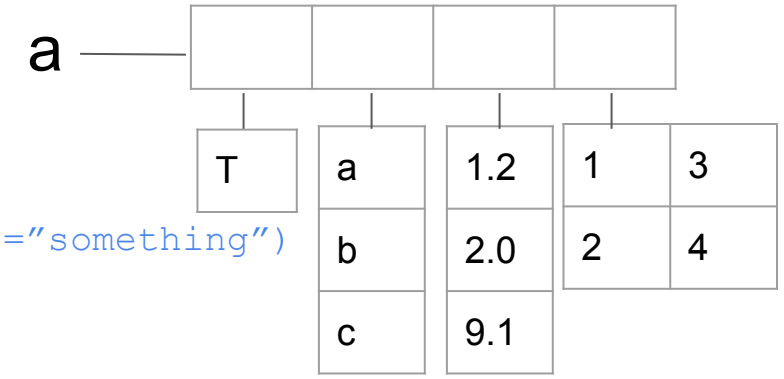
Matrix multiplication extended to tensors (but cf. package ‘tensorA’)

No transposing

No inversion

cf. package ‘abind’ for combining n-dimensional arrays

Data types and structures



Lists

A vector of all kinds of elements:

```
a <- list(name1=c(1,2,3),name2=c(T,F),name3="something")
a
$name1
[1] 1 2 3

$name2
[1]  TRUE FALSE

$name3
[1] "something"
```

Element retrieval by indexing or by name:

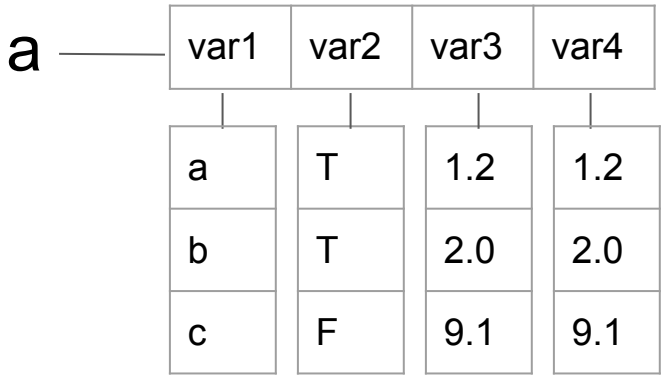
```
a$name1
[1] 1 2 3
a[1]            # but additional level (pointer)
$name1
[1] 1 2 3
a[[1]]          # get actual vector
[1] 1 2 3
a[[1]][2]       # get an element from that vector (a$name1[2] for same
result)
[1] 2
```

Get or set element names using the names function

```
names(a)
```

Data types and structures

Data frames

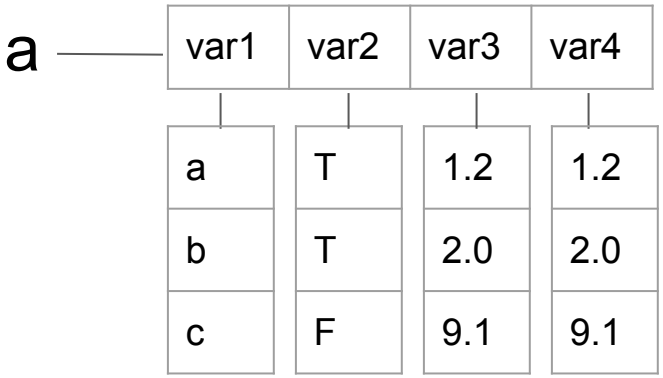


A list of vectors of the same length

```
a <- data.frame(Season = rep(c("Winter","Summer"),each=4),
                Height = rep(c("up","middle","down","profile"),times=2),
                Value  = c(1,4,6,2,9,10,21,3)
               )

a
  Season  Height Value
1 Winter      up     1
2 Winter  middle     4
3 Winter    down     6
4 Winter profile     2
5 Summer      up     9
6 Summer  middle    10
7 Summer    down    21
8 Summer profile     3
```

Data types and structures

Data frames



Can be approached as a list:

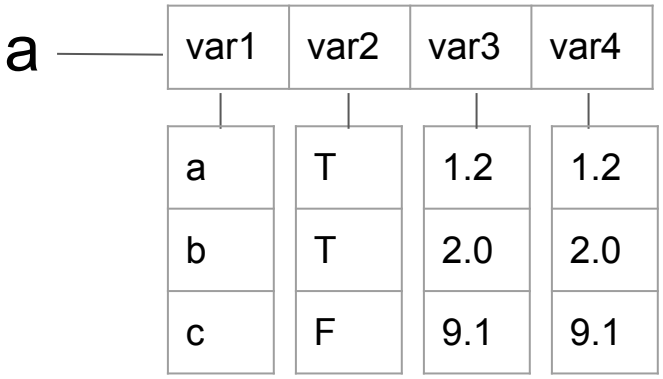
```
names(a)
[1] "Season" "Height" "Value"

a$Season            # character vectors converted to factors  (?factor)
[1] Winter Winter Winter Winter Summer Summer Summer Summer
Levels: Summer Winter

a[[3]][1]
[1] 1
```

Can be approached as a matrix:

```
dim(a)
[1] 8 3

rownames(a)
[1] "1" "2" "3" "4" "5" "6" "7" "8"

a <- cbind(a, data.frame(Quality=rep("good",times=nrow(a))))

head(a,n=3)                          # try also ?tail
  Season Height Value Quality
1 Winter     up     1    good
2 Winter middle     4    good
3 Winter   down     6    good
```

Data types and structures

Data frames



Subsetting

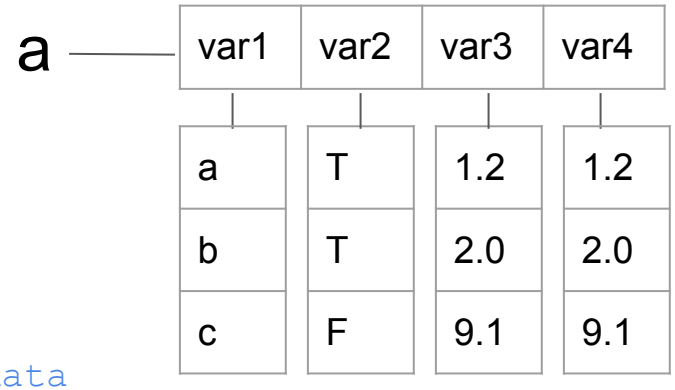
```
b <-subset(a, Season=="Winter" & Height != "middle", select=c(Height,Value))
b
   Height Value
1      up     1
3    down     6
4 profile     2

b$Height          # Height has unused levels (middle)
[1] up   down profile
Levels: down middle profile up

b <- droplevels(b)
b$Height
[1] up   down profile
Levels: down profile up
```

Height is called a factor (integer vector with levels assigned to integer values)

```
Somefactor <- factor(x=rep(1:12,times=2),
                     levels=1:12,
                     labels=c("Jan","Feb","Mar","Apr","May","Jun",
                              "Jul","Aug","Sept","Oct","Nov","Dec")
                    )
```

Data types and structures

Data frames



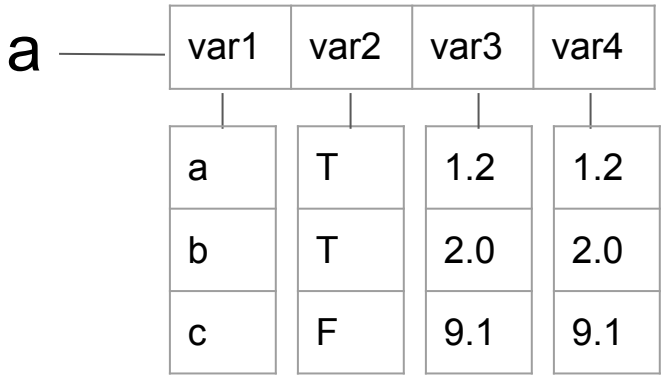
Combining data frames:

```
# suppose we only want winter and summer data

winter <- subset(a, Season==”Winter”)
summer <- subset(a,Season == “Summer”)

winterandsummer <- rbind(winter,summer)
winterandsummer
```

```
  Season  Height Value Quality
1 Winter      up     1    good
2 Winter  middle     4    good
3 Winter    down     6    good
4 Winter profile     2    good
5 Summer      up     9    good
6 Summer  middle    10    good
7 Summer    down    21    good
8 Summer profile     3    good
```

Data types and structures

Data frames



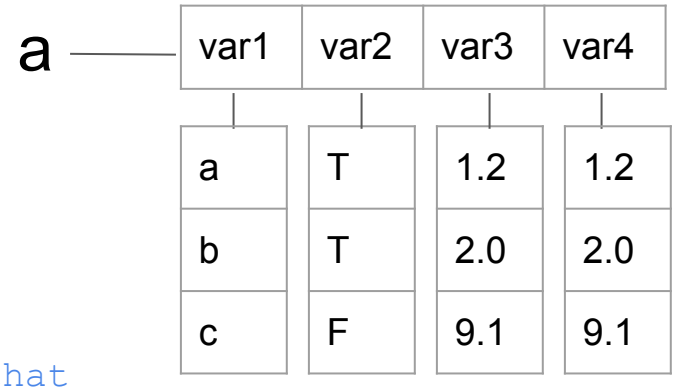
Combining data frames:

```
# suppose we want them next to each other

winterversussummer <- cbind(winter,summer)
winterversussummer
```

```
  Season  Height Value Quality Season  Height Value Quality
1 Winter      up     1    good Summer      up     9     good
2 Winter  middle     4    good Summer  middle    10     good
3 Winter    down     6    good Summer    down    21     good
4 Winter profile     2    good Summer profile     3     good
```

Data types and structures

Data frames

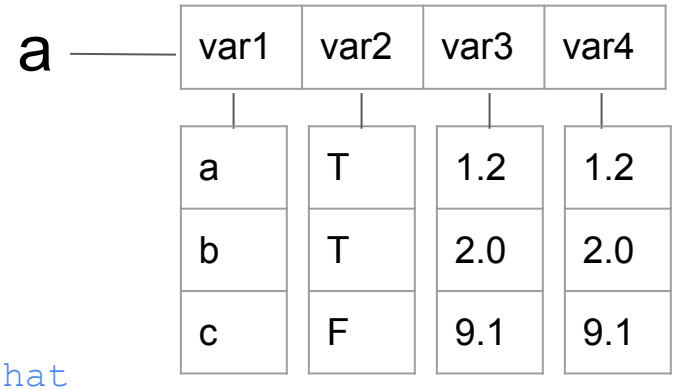


Combining data frames:

```
# suppose we have additional information that
# needs to be added depending on the value of
# one of the variables...

weatherdata <- data.frame (Season=c("Winter","Spring","Summer"),
                           Weather=c("cold","so and so","warm")
                          )
weatherdata
  Season   Weather
1 Winter      cold
2 Spring so and so
3 Summer      warm

alldata <- merge(a,weatherdata,by="Season",all=TRUE)       # full join
  Season Height Value Quality    Weather
1 Summer     up     9    good       warm
2 Summer middle    10    good       warm
...
8 Winter profile    2    good       cold
9 Spring    <NA>   NA    <NA>  so and so
```

Data types and structures

Data frames

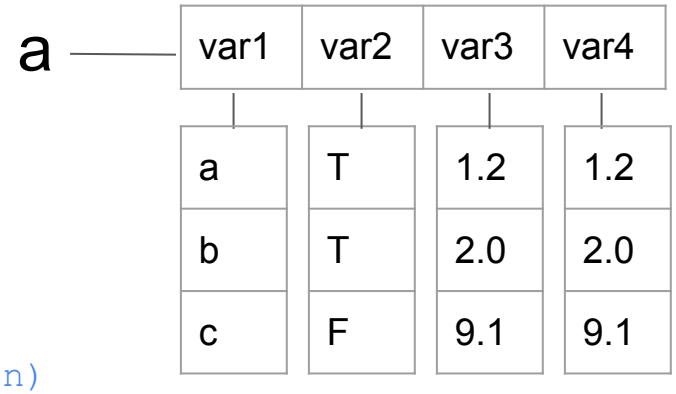


Combining data frames:

```
# suppose we have additional information that
# needs to be added depending on the value of
# one of the variables...

alldata <- merge(a, weatherdata, by="Season", all=FALSE)  # inner join
alldata
  Season  Height Value Quality Weather
1 Summer      up     9    good    warm
2 Summer  middle    10    good    warm
...
7 Winter    down     6    good    cold
8 Winter profile     2    good    cold


# see ?merge for variations to keep the first or second set
# (i.e. left and right outer join)
```

Data types and structures

Data frames

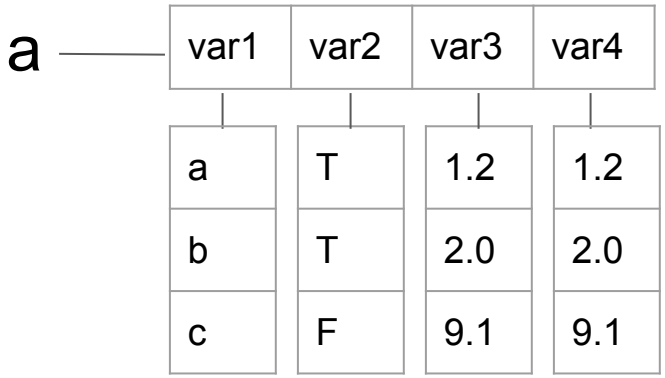


Splitting a data frame using a factor

```
Datasets <- split(alldata, f=alldata$Season)
Datasets
$Summer
  Season  Height Value Quality Weather
1 Summer      up     9    good    warm
2 Summer  middle    10    good    warm
3 Summer    down    21    good    warm
4 Summer profile     3    good    warm

$Winter
  Season  Height Value Quality Weather
5 Winter      up     1    good    cold
6 Winter  middle     4    good    cold
7 Winter    down     6    good    cold
8 Winter profile     2    good    cold
```

Data types and structures

Some useful functions



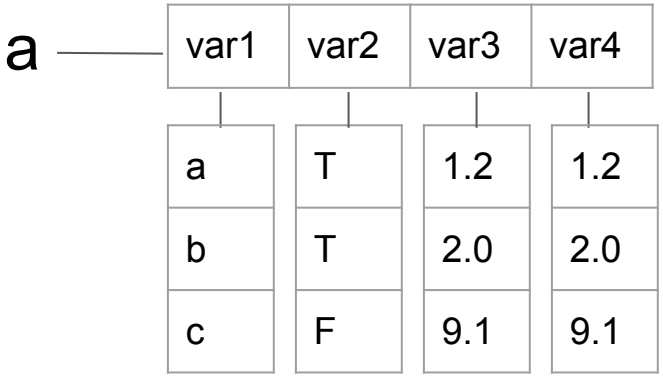
Display internal structure of an object:

```
a <- list(somevector=1:10,
          somematrix=matrix(1:6,ncol=2),
          somelist=list(a=c("a","b","c"),b=c(T,F,F,T)),
          somedataframe=a
         )

str(a)
List of 4
 $ somevector    : int [1:10] 1 2 3 4 5 6 7 8 9 10
 $ somematrix    : int [1:3, 1:2] 1 2 3 4 5 6
 $ somelist  :List of 2
  ..$ a: chr [1:3] "a" "b" "c"
  ..$ b: logi [1:4] TRUE FALSE FALSE TRUE
 $ somedataframe:'data.frame':    8 obs. of  4 variables:
  ..$ Season : Factor w/ 2 levels "Summer","Winter": 2 2 2 2 1 1 1 1
  ..$ Height : Factor w/ 4 levels "down","middle",..: 4 2 1 3 4 2 1 3
  ..$ Value  : num [1:8] 1 4 6 2 9 10 21 3
  ..$ Quality: Factor w/ 1 level "good": 1 1 1 1 1 1 1 1
```

Data types and structures

Some useful functions

Classes and data types




```
a <- list(somevector=1:10,
          somematrix=matrix(1:6,ncol=2),
          somelist=list(a=c("a","b","c"),b=c(T,F,F,T)),
          somedataframe=a
         )

class(a)
[1] "list"

class(a[[1]])
[1] "integer"

class(a[[2]])
[1] "matrix"

> class(a[[3]])
[1] "list"

> class(a[[4]])
[1] "data.frame"
```

Functions

Predefined functions:

- Always the same form: functionname(argument1, argument2, …)
- Argument list can be called by the args-function:

```
args(data.frame)
function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
    fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())

args(mean)
function (x, ...)
```

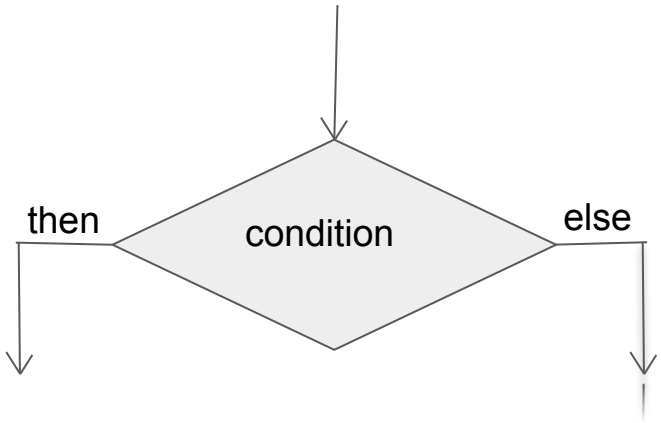
User-defined functions:

```
# an example:
X.squared <- function(x) x^2
x.squared(5)
[1] 25

# an example with code block
pythagoras <- function(x,y) {
    xsq <- x^2; ysq <- y^2
    return(xsq+ysq)          # return exits the function and returns its
argument
}

pythagoras(3,5)
[1] 34
```

Flow control

If … then … else

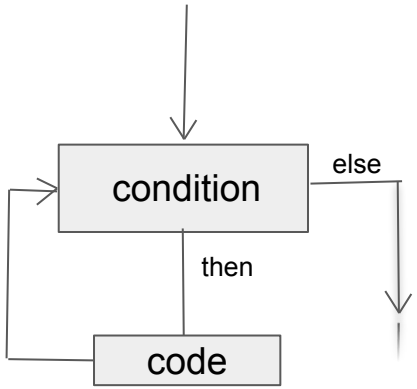
```
# one function per branch
a <- 1
if(a == 1) print("a has value 1") else print("a has not value 1")
[1] "a has value 1"
a <- 2
if(a == 1) print("a has value 1") else print("a has not value 1")
[1] "a has not value 1"

# a code block per branch
a <- 1
if(a == 1) {
     a <- a + 1
     print("a has value 1")
} else {
     a <- a+1
     print("a has not value 1")
}
[1] "a has value 1"
a
[1] 2
if(a == 1) {
     a <- a + 1
     print("a has value 1")
} else {
     a <- a+1
     print("a has not value 1")
}
[1] "a has not value 1"
```



Flow control

Loops

```
# for loop
for(i in 1:10) print(i)

Months <- c("Jan","Feb","Mar","Apr","Jun","Jul","Aug","Sep","Oct","Nov","Dec")
for(month in months) print(month)

# while loop
i <- 1
while(i <= 12) {
  print(Months[i])
  i <- i+1
}
```



More optimized ways of looping:

- apply
- lapply
- sapply
- ...

Flow control

- apply

```
Mat <- cbind(1:10,1:10,1:10)
Mat
      [,1] [,2] [,3]
 [1,]    1    1    1
 [2,]    2    2    2
 [3,]    3    3    3
 [4,]    4    4    4
 [5,]    5    5    5
 [6,]    6    6    6
 [7,]    7    7    7
 [8,]    8    8    8
 [9,]    9    9    9
[10,]   10   10   10

apply(Mat,MAR=1,sum)                    # look up  ?sum
[1]  3  6  9 12 15 18 21 24 27 30

apply(Mat,MAR=2,sum)
[1] 55 55 55
```

Flow control

- lapply

```
SomeList <- list(a=1:10,b=seq(0,1,by=0.01),c=rep(5,times=20))

str(SomeList)
List of 3
 $ a: int [1:10] 1 2 3 4 5 6 7 8 9 10
 $ b: num [1:101] 0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09
...
 $ c: num [1:20] 5 5 5 5 5 5 5 5 5 5 ...


lapply(SomeList,FUN=mean )
$a
[1] 5.5

$b
[1] 0.5

$c
[1] 5

lapply(SomeList,FUN=function(x)
     {
       res <- data.frame(n=length(x),mean=mean(x),sd=sd(x))
       return(res)
     }
)
$a
```

Flow control

- lapply

```
# with a user-defined function and result stored in a variable …

Result <- lapply(SomeList,FUN=function(x)
     {
       res <- data.frame(n=length(x),mean=mean(x),sd=sd(x))
       return(res)
     }
)

Result
$a
   N mean       sd
1 10  5.5 3.02765

$b
    n mean         sd
1 101  0.5 0.2930017

$c
   n mean sd
1 20    5  0

Result$a[“mean”]
  mean
1  5.5
```

Flow control

- lapply

```
# find number of observations in alldata (split up in Datasets) per
# season that are not at the middle height ...

Result <- lapply(Datasets,FUN=function(x) {
                                Notmiddle
<-subset(x,Height!=”middle”)
                                Nobs <- nrow(Notmiddle)
                                return(Nobs)
                                }
                )

Result
$Summer
[1] 3

$Winter
[1] 3
```

R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07

Getting help

Most R functions have online documentation.

help(topic) documentation on topic
?topic id.
help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
str(a) display the internal *str*ucture of an R object
summary(a) gives a "summary" of a, usually a statistical summary but it is *generic* meaning it has different operations for different classes of a
ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of a
methods(class=class(a)) lists all the methods to handle objects of class a

Input and output

load() load the datasets written with save
data(x) loads specified data sets
library(x) load add-on packages
read.table(file) reads a file in table format and creates a data frame from it; the default separator sep="" is any whitespace; use header=TRUE to read the first line as a header of column names; use as.is=TRUE to prevent character vectors from being converted to factors; use comment.char="" to prevent "#" from being interpreted as a comment; use skip=n to skip n lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename",header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename",header=TRUE) id. but with defaults set for reading tab-delimited files
read.fwf(file,widths,header=FALSE,sep="",as.is=FALSE) read a table of *f*ixed *width f*ormatted data into a 'data.frame'; widths is an integer vector, giving the widths of the fixed-width fields
save(file,...) saves the specified objects (...) in the XDR platform-independent binary format
save.image(file) saves all objects
cat(..., file="", sep=" ") prints the arguments after coercing to character; sep is the character separator between arguments
print(a, ...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ") prints x after converting to a data frame; if quote is TRUE, character or factor columns are surrounded by quotes ("); sep is the field separator; eol is the end-of-line separator; na is the string for missing values; use col.names=NA to add a blank column header to get the column headers aligned correctly for spreadsheet input
sink(file) output to file, until sink()

Most of the I/O functions have a file argument. This can often be a character string naming a file or a connection. file="" means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with description = "clipboard". To read a table copied from Excel, use
x <- read.delim("clipboard")
To write a table to the clipboard for Excel, use
write.table(x,"clipboard",sep="\t",col.names=NA)
For database interaction, see packages RODBC, DBI, RMySQL, RPgSQL, and ROracle. See packages XML, hdf5, netCDF for reading other file formats.

Data creation

c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector
from:to generates a sequence; ":" has operator priority; 1:4 + 1 is "2,3,4,5"
seq(from,to) generates a sequence by= specifies increment; length= specifies desired length
seq(along=x) generates 1, 2, ..., length(along); useful for for loops
rep(x,times) replicate x times; use each= to repeat "each" element of x each times; rep(c(1,2,3),2) is 1 2 3 1 2 3; rep(c(1,2,3),each=2) is 1 1 2 2 3 3
data.frame(...) create a data frame of the named or unnamed arguments; data.frame(v=1:4,ch=c("a","B","c","d"),n=10); shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi",c=3i);
array(x,dim=) array with data x; specify dimensions like dim=c(3,4,2); elements of x recycle if x is not long enough
matrix(x,nrow=,ncol=) matrix; elements of x recycle
factor(x,levels=) encodes a vector x as a factor
gl(n,k,length=n*k,labels=1:n) generate levels (factors) by specifying the pattern of their levels; k is the number of levels, and n is the number of replications
expand.grid() a data frame from all combinations of the supplied vectors or factors
rbind(...) combine arguments by rows for matrices, data frames, and others
cbind(...) id. by columns

Slicing and extracting data

Indexing vectors

x[n]	n^{th} element
x[-n]	all *but* the n^{th} element
x[1:n]	first n elements
x[-(1:n)]	elements from n+1 to the end
x[c(1,4,2)]	specific elements
x["name"]	element named "name"
x[x > 3]	all elements greater than 3
x[x > 3 & x < 5]	all elements between 3 and 5
x[x %in% c("a","and","the")]	elements in the given set

Indexing lists

x[n]	list with elements n
x[[n]]	n^{th} element of the list
x[["name"]]	element of the list named "name"
x$name	id.

Indexing matrices

x[i,j]	element at row i, column j
x[i,]	row i
x[,j]	column j
x[,c(1,3)]	columns 1 and 3
x["name",]	row named "name"

Indexing data frames (matrix indexing plus the following)

x[["name"]]	column named "name"
x$name	id.

Variable conversion

as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information

is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)
length(x) number of elements in x
dim(x) Retrieve or set the dimension of an object; dim(x) <- c(3,2)
dimnames(x) Retrieve or set the dimension names of an object
nrow(x) number of rows; NROW(x) is the same but treats a vector as a one-row matrix
ncol(x) and **NCOL(x)** id. for columns
class(x) get or set the class of x; class(x) <- "myclass"
unclass(x) remove the class attribute of x
attr(x,which) get or set the attribute which of x
attributes(obj) get or set the list of attributes of obj

Data selection and manipulation

which.max(x) returns the index of the greatest element of **x**
which.min(x) returns the index of the smallest element of **x**
rev(x) reverses the elements of x
sort(x) sorts the elements of x in increasing order; to sort in decreasing order: rev(sort(x))
cut(x,breaks) divides x into intervals (factors); breaks is the number of cut intervals or a vector of cut points
match(x, y) returns a vector of the same length than x with the elements of x which are in y (NA otherwise)
which(x == a) returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of i for which x[i] == a (the argument of this function must be a variable of mode logical)
choose(n, k) computes the combinations of k events among n repetitions $= n!/[(n-k)!k!]$
na.omit(x) suppresses the observations with missing data (NA) (suppresses the corresponding line if x is a matrix or a data frame)
na.fail(x) returns an error message if x contains at least one NA

...

ScienceR:

Beginners course for R users on the Github repository

If you have questions on the course material, ask them in the issues section of the repo with label “ScienceR”