An introduction to 

Data types and structures

Noémie Becker & Benedikt Holtmann

Winter Semester 16/17

Course outline

- Review – Getting started with R
- Creating Objects
- Data types in R
- Data structures in R
  - Vectors
  - Matrices
  - Data frames
- Accessing data

Review – Getting started with R

Important commands and functions	
install.packages(*packagename*)	installs packages
library(*packagename*)	loads packages into R environment
setwd("~/Desktop")	sets working directory
getwd()	gets working directory
help("mean") ?mean	when you need help
example("mean")	examples for the usage of 'mean'
#	to comment your script R ignores everything to the right

Creating objects

General form:

variable <- value

variable = value
variable -> value

Examples:

x <- 3 # The variable ‘x’ is assigned the value ‘3’

Creating objects

General form:

variable <- value

Examples:

x <- 3 # The variable ‘x’ is assigned the value ‘3’

Works with longer expressions

```
y <- x^2 +3
y
[1] 12
```

Creating objects

General form:

variable <- value

Examples:

x <- 3 # The variable ‘x’ is assigned the value ‘3’

Works with longer expressions or to define functions

```
y <- x^2 +3
y
[1] 12
```

```
MyFunction <- sqrt
MyFunction(81)
[1] 9
```

Creating objects

General form:

variable <- value

Allowed variable /object names:
my.variable, my_variable, myVariable, a, b, x1, x2, data2, 2data

Not allowed: '.' followed by a number at the beginning
.4you

and neither are the reserved words, e.g: if, else, repeat, while function, for, FALSE, TRUE, etc.

Data types and structures

The terms **‘type’** and **‘structure’** are often used interchangeable!

Types:

- logical
- numeric
- integer
- character
- complex

Structures:

- Vector
- Factor
- List
- Matrix
- Data frame

Data types

Data type	Description	Example
logical	True or False	TRUE, FALSE
numeric double	real numbers or decimals	2.3, pi, sqrt(2)
integer	whole numbers	-5, 1, 7 in R: -5L, 1L, 7L
character	character or string	“male”, “female”
complex	complex numbers	2.1+3i, 1+0i

Vectors

- A collection of values that all have the same data type
- One-dimensional

Examples: (-2, 3.4, 3.75, 5.2, 6)

(TRUE, FALSE, TRUE, TRUE, FALSE)

(“blue”, “green”, “yellow”, “red”)

Vectors

Vectors can be created using the following **functions**:

c() function to combine individual values

seq() to create more complex sequences

rep() to create replicates of values

Vectors

Examples of c()

```
c(2,7,8,12,3,25)
[1]  2  7  8 12  3 25

c(2:5, 3:6)
[1] 2 3 4 5 3 4 5 6
```

```
x <- c(2,7,8,12,3,25)
x
[1]  2  7  8 12  3 25
```

```
y <- c(red="Bob", blue="Dave", green="Jenny")
y
   red       blue      green
 "Bob"  "Dave"   "Jenny"
```

Vectors

Examples of seq()

```
seq(from=1, to=8)
[1] 1 2 3 4 5 6 7 8

seq(from=4, to=10, by=2)
[1]  4  6  8 10

seq(from=1, to=10, length=4)
[1]  1  4  7 10

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

Vectors

Examples of seq()

```
seq(from=1, to=8)
[1] 1 2 3 4 5 6 7 8
```

```
seq(1, 8)
[1] 1 2 3 4 5 6 7 8
```

```
seq(from=4, to=10, by=2)
[1]  4  6  8 10
```

```
seq(4, 10, 2)
[1]  4  6  8 10
```

```
seq(from=1, to=10, length=4)
[1]  1  4  7 10
```

```
seq(1, 10,   , 4)
[1]  1  4  7 10
```

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

Vectors

Examples of rep()

```
rep(1, 4)
[1] 1 1 1 1

rep(4:5, 3)
[1] 4 5 4 5 4 5

rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4

rep(1:4, times=2, each=2)
[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

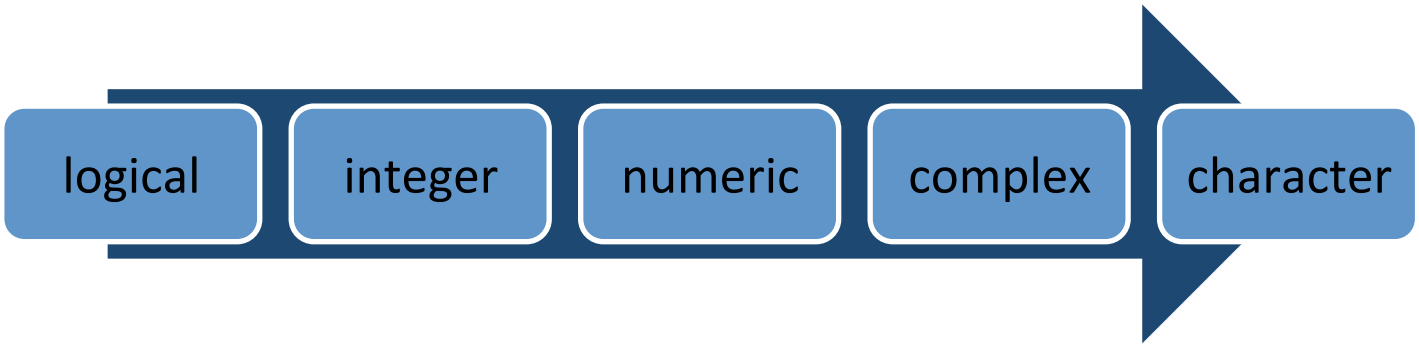
Vectors

Data type conversion

Vectors may only have one type

When combining different types, R will **coerce** a vector to the most flexible type

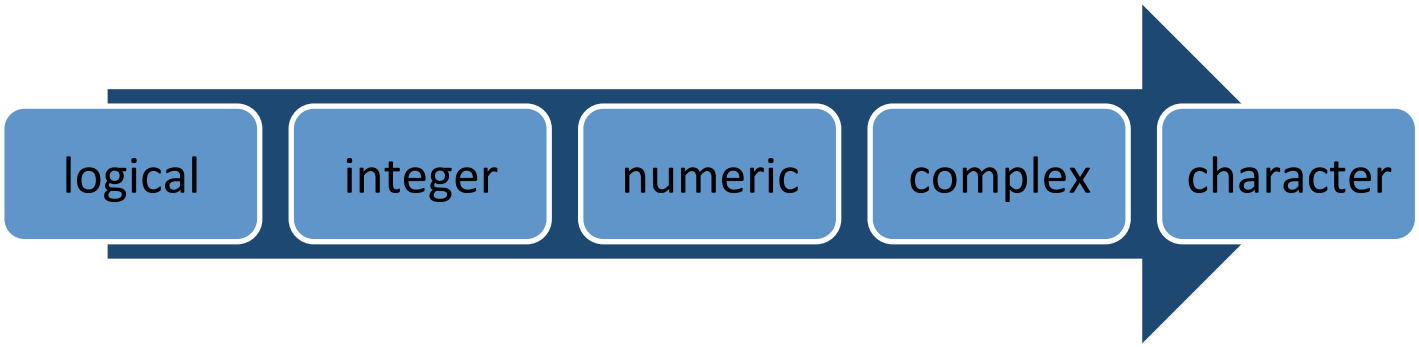
Coercion rule in R:



Vectors

Examples

```
x <- c(5, "b")                # becomes a character
[1] "5" "b"                   # " " indicate characters

y <- c(FALSE, 3)              # becomes a numeric
[1] 0  3
```



Vectors

Coercion

You also can coerce vectors using the “as.class_name” function

Examples: as.numeric(x)
as.logical(x)
as.integer(x)
as.character(x)
as.complex(x)
as.factor(x)

You can check the type or class of a vector using **typeof()** or **class()**

Operations on vectors

You access elements of a vector with the [ ]-Operator:

```
x <- c(2, 4, 6, 8, 10)
x[4]
[1] 8

x[3:5]
[1]  6  8 10

x[-2]
[1]  2  6  8 10
```

Operations on vectors

Standard operations on vectors are element by element:

```
c(2, 5, 3) + c(4, 2, 7)
[1]  6   7  10

2 + c(2, 5, 3)
[1] 4  7  5

c(2, 5, 3)^2
[1]  4  25   9
```

Vectors

Other useful functions	
class(*vector name*)	Returns class/type of a vector
length(*vector name*)	Returns the total number of elements
x[length(x)]	Returns last value of a vector
rev(*vector name*)	Returns the reversed vector
sort(*vector name*)	Returns the sorted vector
unique(*vector name)*	Returns vector without multiple elements

Factors

- A factor is a vector that represents categorical data
- Can only contain predefined categories
- Can be ordered and unordered

Examples: ("yes", "no", "no", "yes", "yes")

("male", "female", "female", "male")

("small", "large", "small", "medium")

Factors

Factors can be created using **factor()**:

```
size <- factor(c("small", "large", "small", "medium"))

size

[1] small  large  small  medium
Levels: large medium small          # unordered factor
```

Note: Quotes, such as "male" are not shown and levels are printed

Factors

Factors can be created using **factor()**:

```
size <- factor(c("small", "large", "small", "medium"),
                        levels = c("small", "medium", "large"))

size

[1] small  large  small  medium
Levels: small medium large          # ordered factor
```

The levels of an factor can be displayed using **levels()**

Lists

- A collection of data structures
- A list can encompass any data types, including lists
- Objects can have different lengths
- You can construct lists by using **list()**
- Almost all functions (e.g., t-test, linear regression, etc.) in R produce output that is stored in a list

Lists

Example of a list:

```
myList <- list(1:3, c("a", "b"), c(TRUE, FALSE, TRUE))
str(myList)

List of 3
 $ : int [1:3] 1 2 3
 $ : chr [1:2] "a" "b"
 $ : logi [1:3] TRUE FALSE TRUE
```

Matrix

- A collection of values that all have the same data type
- Two-dimensional, arranged in **rows** and **columns**

Matrices

- A collection of values that all have the same data type
- Two-dimensional, arranged in **rows** and **columns**

Example:

$$\begin{pmatrix} 1 & 7 & 12 \\ 23 & 8 & -3 \\ 4 & -9 & 5 \end{pmatrix}$$

3-by-3 matrix

Matrices

Matrices can be created using the **functions**:

matrix()	creates a matrix by specifying rows and columns
dim()	sets dimensions to a vector
cbind() **rbind()**	combines columns or rows

Matrices

Example of matrix()

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = FALSE)

matrix(
    data = 1:6,          # the data elements
    nrow = 2,            # number of rows
    ncol = 3),           # number of columns
    byrow = FALSE)       # fill matrix by row
```

Matrices

Example of matrix()

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = FALSE)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
matrix(1:6, 2, 3)    # is the same
```

Matrices

Example of dim()

```
x <- 1:6
dim(x) <- c(2, 3)
x
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
y <- 1:6
dim(y) <- c(3, 2)
y
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Note: **dim()** can also be used to retrieve dimensions of an object!

Matrices

Examples of cbind() and rbind()

```
x <- c(1, 2, 3)
y <- 10:12
```

```
cbind(x, y)
```

```
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
```

```
rbind(x, y)
```

```
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Matrices

Assign names to rows and columns of a matrix

```
z<- matrix(1:6, 2, 3)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
rownames(z) <- c("A", "B")
colnames(z) <- c("a", "b", "c")
z
```

```
  a b c
A 1 3 5
B 2 4 6
```

Data frames

- A collection of vectors that are of equal length
- Two-dimensional, arranged in **rows** and **columns**
- Columns can contain vectors of different data types
  - **but** within a column, every cell must be the same type of data
- Used to represent entire data sets

Data frames

Example

Bird_ID	Sex	Mass	Wing
Bird_1	F	17.45	75.0
Bird_2	F	18.20	75.0
Bird_3	M	18.45	78.25
Bird_4	F	17.36	74.4
Bird_5	M	18.90	84.0
Bird_6	M	19.16	81.83

- Usually each column is named → variables
- Rows depict different observations, measurements

Data frames

Data frames can be created using the **function**:

data.frame() # creates a data frame object from a set of vectors

Data frames

Example of data.frame()

```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))
```

```
df <- data.frame(
        ID = 1:3,                    # data elements first column
        Sex = c("F", "F", "M"),      # data elements second column
        Mass = c(17, 18, 18))        # data elements third column
                                     # column names
```

Note:

Columns must be of same length

R uses the equal sign to specify named arguments

Data frames

Example of data.frame()

```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))
```

	ID	Sex	Mass
1	1	F	17
2	2	F	18
3	3	M	18

Data frames

Important: data.frame() automatically turns strings into factors

```
df_2 <- data.frame(x = 1:3, y = c("a", "b", "c"))

str(df_2)      # str() displays internal structure of an R object

'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Argument stringsAsFactors = FALSE prevents this behaviour

Data frames

Important: data.frame() automatically turns strings into factors

```
df_2 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)

str(df_2)

'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr w/ 3 levels "a","b","c"
```

Data frames

- Creating a data frame by hand takes a lot of time
- Also, typing invites typos and errors
- You should avoid typing large data sets into R by hand

Data frames

- Creating a data frame by hand takes a lot of time
- Also, typing invites typos and errors
- You should avoid typing large data sets into R by hand

➡ Import entire data sets into R

Usually data frames are imported using the functions:
read.csv() or **read.table()**

More next time!

Accessing data

Indexing by integer vector

You can use **x[ ]** to look up a single element or multiple elements in a vector

```
x<- (10:22)
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22
```

```
x[7]
[1] 16
```

```
x[1:4]
[1] 10 11 12 13
```

```
x[c(1,4,9,12)]
[1] 10 13 18 21
```

Accessing data

Indexing by integer vector

You can also use **negative integers** to return a vector consisting of all elements except the specified elements:

```
x[-2:-7]                                  # excludes elements 2 to 7
[1] 10 17 18 19 20 21 22
```

Accessing data

Indexing by integer vector

In multidimensional data structures (e.g. matrices and data frames) an element at the m^{th} row, n^{th} column can be accessed by the expression **x[m, n]**

Accessing data

Indexing by integer vector

```
z<- matrix(data=c(10:21), nrow = 3, ncol = 4)

z
     [,1] [,2] [,3] [,4]
[1,]   10   13   16   19
[2,]   11   14   17   20
[3,]   12   15   18   21

z[2, 3]          # indexing is row by column
[1] 17
```

Accessing data

Indexing by integer vector

The entire m^{th} row can be extracted by the expression **x[m,]**

```
z<- matrix(data=c(10:21), nrow = 3, ncol = 4)

z[2, ]

[1] 11 14 17 20
```

Accessing data

Indexing by integer vector

The entire n^{th} column can be extracted by the expression **x[, n]**

```
z<- matrix(data=c(10:21), nrow = 3, ncol = 4)

z[ , 3]

[1] 16 17 18
```

Accessing data

Indexing by integer vector

Multiple rows or columns can be extracted by:

```
z<- matrix(data=c(10:21), nrow = 3, ncol = 4)
```

```
z[1:2, 1:2]
```

```
     [,1] [,2]
[1,]   10   13
[2,]   11   14
```

```
z[1:2, c(1, 3)]
```

```
     [,1] [,2]
[1,]   10   16
[2,]   11   17
```

Accessing data

Indexing by name

You can index an element by name using the $ notation

```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))

str(df)
```

```
'data.frame':   3 obs. of  3 variables:
$ ID  : int  1 2 3
$ Sex : Factor w/ 2 levels "F","M": 1 1 2
$ Mass: num  17 18 18
```

Accessing data

Indexing by name

You can index an element by name using the $ notation

```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))

df$Mass

[1] 17 18 18
```

Accessing data

Indexing by name

You can also use the single-bracket notation [ ] to index a set of elements by name

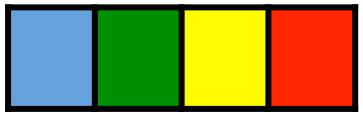
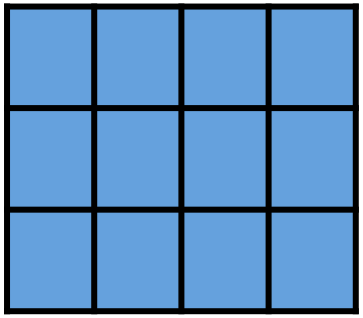
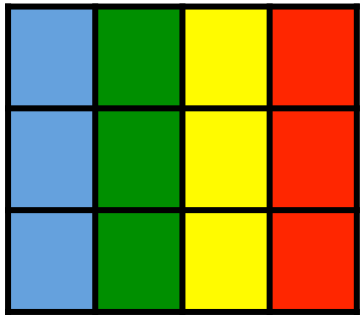
```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))

df[c("Sex", "Mass")]
```

```
  Sex Mass
1   F   17
2   F   18
3   M   18
```

Summary of R data types

```
example_logical <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

example_integer <- c(1L, 2L, 7L, 5L, 9L)

example_numeric <- c(0, 2.1, 2.75, 3.99, -1.5)

exaple_character <- c("apostrophes" , "indicate", "characters")

eample_factor <- factor(c("small", "large", "large", "medium"))
```

Summary of R data structures

	Single type	multiple types
1D	Vector 	List 
2D	Matrix 	Data frame 

Which R functions did we learn?

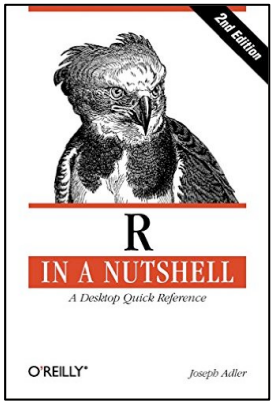
<-	assigns the value to an object
=	used to specify values of arguments within functions
c()	concatenates objects to create new vectors
:	used to create a sequence of numbers
seq()	generates sequences
rep()	replicates elements of vectors
as.numeric() as.integer() as.character() etc.	makes sure that all values of a given vector are numeric, integer, character, etc.
class()	prints the class of a vector
length()	gets or sets the length of a vector
rev()	provides a reversed version of its argument
sort()	sorts a vector into ascending or descending order

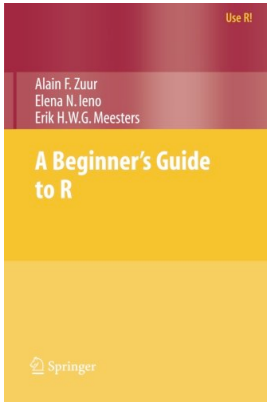
Which R functions did we learn?

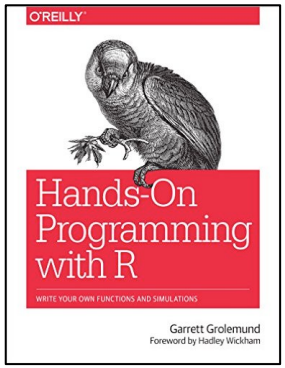
unique()	removes duplicate elements or rows from a vectors, matrices or data frames
levels()	returns the levels of an object
matrix()	creates a matrix from a given set of values
dim()	retrieve or set the dimension of an object
cbind() & rbind()	combines vectors by columns or rows
rownames() & colnames()	get and set the row or column names of matrices or data frames
data.frame()	creates a data frame from specified data
str()	shows the structure of an object
x[i] & x[m, n]	operator to extract or replace values of vectors, matrices or data frames by integer or name
$	operator to access or replace values of matrices or data frames by name

Further reading

Books







Internet

Environmental Computing

Advanced R by Hadley Wickham