An introduction to R

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
### Important commands and functions

<table>
<thead>
<tr>
<th>Command/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>install.packages()</td>
<td>installs packages</td>
</tr>
<tr>
<td>library()</td>
<td>loads packages into R environment</td>
</tr>
<tr>
<td>setwd(&quot;~/Desktop&quot;)</td>
<td>sets working directory</td>
</tr>
<tr>
<td>getwd()</td>
<td>gets working directory</td>
</tr>
<tr>
<td>help(&quot;mean&quot;)</td>
<td>when you need help</td>
</tr>
<tr>
<td>?mean</td>
<td>examples for the usage of ‘mean’</td>
</tr>
<tr>
<td>example(&quot;mean&quot;)</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>to comment your script</td>
</tr>
<tr>
<td></td>
<td>R ignores everything to the right</td>
</tr>
</tbody>
</table>
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

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

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

**Types:**
- logical
- numeric
- integer
- character
- complex

**Structures:**
- Vector
- Factor
- List
- Matrix
- Data frame
## Data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical</td>
<td>True or False</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>numeric</td>
<td>real numbers or decimals</td>
<td>2.3, pi, sqrt(2)</td>
</tr>
<tr>
<td>double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>whole numbers</td>
<td>-5, 1, 7</td>
</tr>
<tr>
<td></td>
<td>in R: -5L, 1L, 7L</td>
<td></td>
</tr>
<tr>
<td>character</td>
<td>character or string</td>
<td>“male”, “female”</td>
</tr>
<tr>
<td>complex</td>
<td>complex numbers</td>
<td>2.1+3i, 1+0i</td>
</tr>
</tbody>
</table>
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
Examples of `c()`

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

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

```r
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
Data types and structures

Vectors

Examples of seq()

```r
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
```
Data types and structures

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
Data types and structures

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:

- logical
- integer
- numeric
- complex
- character
Data types and structures

Vectors

Examples

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

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

```r
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:

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

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

\[
c(2, 5, 3)^2
\]
\[
[1] 4 25 9
\]
### Data types and structures

#### Vectors

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>class(vector name)</code></td>
<td>Returns class/type of a vector</td>
</tr>
<tr>
<td><code>length(vector name)</code></td>
<td>Returns the total number of elements</td>
</tr>
<tr>
<td><code>x[length(x)]</code></td>
<td>Returns last value of a vector</td>
</tr>
<tr>
<td><code>rev(vector name)</code></td>
<td>Returns the reversed vector</td>
</tr>
<tr>
<td><code>sort(vector name)</code></td>
<td>Returns the sorted vector</td>
</tr>
<tr>
<td><code>unique(vector name)</code></td>
<td>Returns vector without multiple elements</td>
</tr>
</tbody>
</table>
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 can be created using `factor()`:

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

```r
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 can be created using `factor()`:

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

```r
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
Data types and structures

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:

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>matrix()</code></td>
<td>creates a matrix by specifying rows and columns</td>
</tr>
<tr>
<td><code>dim()</code></td>
<td>sets dimensions to a vector</td>
</tr>
<tr>
<td><code>cbind()</code></td>
<td>combines columns or rows</td>
</tr>
<tr>
<td><code>rbind()</code></td>
<td></td>
</tr>
</tbody>
</table>
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()**

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

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

```r
matrix(1:6, 2, 3)  # is the same
```
Example of `dim()`

```r
x <- 1:6
[1:6]
[,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6

y <- 1:6
[1:6]
[,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()`

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

cbind(x, y)  # Output:
             [,1] [,2] [,3]
[1,]    1    10   NA
[2,]    2    11   NA
[3,]    3    12   NA

rbind(x, y) # Output:
             [,1] [,2] [,3]
[1,]    1    2    3
[2,]    10   11   12
```

Data types and structures
### Matrices

**Assign names** to rows and columns of a matrix

```r
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")
```

```
     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

<table>
<thead>
<tr>
<th>Bird_ID</th>
<th>Sex</th>
<th>Mass</th>
<th>Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bird_1</td>
<td>F</td>
<td>17.45</td>
<td>75.0</td>
</tr>
<tr>
<td>Bird_2</td>
<td>F</td>
<td>18.20</td>
<td>75.0</td>
</tr>
<tr>
<td>Bird_3</td>
<td>M</td>
<td>18.45</td>
<td>78.25</td>
</tr>
<tr>
<td>Bird_4</td>
<td>F</td>
<td>17.36</td>
<td>74.4</td>
</tr>
<tr>
<td>Bird_5</td>
<td>M</td>
<td>18.90</td>
<td>84.0</td>
</tr>
<tr>
<td>Bird_6</td>
<td>M</td>
<td>19.16</td>
<td>81.83</td>
</tr>
</tbody>
</table>

• Usually each column is named variables

• Rows depict different observations, measurements
Data frames can be created using the function:

```r
data.frame()  # creates a data frame object from a set of vectors
```
Data frames

Example of `data.frame()`

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

```r
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
Example of `data.frame()`

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Sex</th>
<th>Mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>18</td>
</tr>
</tbody>
</table>
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
Important: data.frame() automatically turns strings into factors

```r
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()
Indexing by integer vector

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

```r
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:

```r
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\textsuperscript{th} row, n\textsuperscript{th} column can be accessed by the expression \texttt{x[m, n]}
## Accessing data

### Indexing by integer vector

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

z

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>10</td>
<td>13</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>[2,]</td>
<td>11</td>
<td>14</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>[3,]</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
</tr>
</tbody>
</table>
```

```r
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 $x[m, ]$

```r
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 $x[, n]$

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

z[, 3]

[1] 16 17 18
```
Indexing by integer vector

Multiple rows or columns can be extracted by:

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

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

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

```
[,1] [,2]
[1,] 10  16
[2,] 11  17
```
Indexing by name

You can index an element by name using the $ notation.

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

```
df: 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
Indexing by name

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Sex</th>
<th>Mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>18</td>
</tr>
</tbody>
</table>
Summary of R data types

eexample_logical <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

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

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

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

eexample_factor <- factor(c("small", "large", "large", "medium"))
Summary of R data structures

Single type
- Vector
  - 1D
  - 2D

Multiple types
- List
- Data frame
## Which R functions did we learn?

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-</td>
<td>assigns the value to an object</td>
</tr>
<tr>
<td>=</td>
<td>used to specify values of arguments within functions</td>
</tr>
<tr>
<td>c()</td>
<td>concatenates objects to create new vectors</td>
</tr>
<tr>
<td>:</td>
<td>used to create a sequence of numbers</td>
</tr>
<tr>
<td>seq()</td>
<td>generates sequences</td>
</tr>
<tr>
<td>rep()</td>
<td>replicates elements of vectors</td>
</tr>
<tr>
<td>as.numeric()</td>
<td>makes sure that all values of a given vector are numeric, integer, character, etc.</td>
</tr>
<tr>
<td>as.integer()</td>
<td></td>
</tr>
<tr>
<td>as.character()</td>
<td></td>
</tr>
<tr>
<td>class()</td>
<td>prints the class of a vector</td>
</tr>
<tr>
<td>length()</td>
<td>gets or sets the length of a vector</td>
</tr>
<tr>
<td>rev()</td>
<td>provides a reversed version of its argument</td>
</tr>
<tr>
<td>sort()</td>
<td>sorts a vector into ascending or descending order</td>
</tr>
</tbody>
</table>
## Which R functions did we learn?

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unique()</code></td>
<td>Removes duplicate elements or rows from a vectors, matrices or data frames</td>
</tr>
<tr>
<td><code>levels()</code></td>
<td>Returns the levels of an object</td>
</tr>
<tr>
<td><code>matrix()</code></td>
<td>Creates a matrix from a given set of values</td>
</tr>
<tr>
<td><code>dim()</code></td>
<td>Retrieve or set the dimension of an object</td>
</tr>
<tr>
<td><code>cbind()</code> &amp; <code>rbind()</code></td>
<td>Combines vectors by columns or rows</td>
</tr>
<tr>
<td><code>rownames()</code> &amp; <code>colnames()</code></td>
<td>Get and set the row or column names of matrices or data frames</td>
</tr>
<tr>
<td><code>data.frame()</code></td>
<td>Creates a data frame from specified data</td>
</tr>
<tr>
<td><code>str()</code></td>
<td>Shows the structure of an object</td>
</tr>
<tr>
<td><code>x[i]</code> &amp; <code>x[m, n]</code></td>
<td>Operator to extract or replace values of vectors, matrices or data frames by integer or name</td>
</tr>
<tr>
<td><code>$</code></td>
<td>Operator to access or replace values of matrices or data frames by name</td>
</tr>
</tbody>
</table>
Further reading

Books

- R in a Nutshell
- A Beginner's Guide to R
- Hands-On Programming with R
- Advanced R by Hadley Wickham

Internet

- Environmental Computing
- R Tutorial: An R Introduction to Statistics
- RBloggers