
```
roundedpi <- 3,14
radius <- 20
circle <- 2*roundedpi*radius
```

Where is the error?

Answer: sign for decimal is , instead of .

An introduction to R
WS 2019/2020

Data types and structures

Dr. Noémie Becker
Dr. Eliza Argyridou

Special thanks to:
Prof. Dr. Martin Hutzenthaler, Dr. Sonja Grath and Dr. Benedikt Holtmann for significant contributions to course development, lecture notes and exercises

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

Data types and structures

Examples:

Data types

- logical
- double/numeric
- integer
- character
- complex

Structures

- vector
- factor
- list
- matrix
- data frame

R is an object-oriented language:
- Every object in R has a *type*.
- Every object in R is member of a *class*.

To determine the class of an object:
`class()`
To determine the underlying structure of an object:
`typeof()`

Data types in R

Data type	Description	Examples
logical	TRUE or FALSE	TRUE, FALSE
integer	whole numbers	-5L, 1L, 7L
numeric	integers and real numbers	5, -2, 3.1415, sqrt(2)
complex	complex numbers	2.1+3i, 5+0i
character	character string	“This is text”, “5”

Types can be *explicitly* converted:
`as.logical(), as.integer(), as.numeric()/as.double, as.complex(), as.character()`

You can check for a data type:
`is.logical(), is.integer(), is.numeric()/is.double, is.complex(), is.character()`

Data types in R

Internal representation of TRUE and FALSE in R:

```
> as.integer(TRUE)
[1] 1
> as.integer(FALSE)
[1] 0
```


```
x <- 5
is.integer(x)
is.double(x)
x <- 5L
is.integer(x)
is.double(x)
```

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

What is a vector?

The "smallest" data structure in R is a *vector*.

- A vector is a collection of values that all have the same data type
- A vector is one-dimensional

Examples:
(-2, 3.4, 3.75, 5.2, 6)
(TRUE, FALSE, TRUE, TRUE, FALSE)
("blue", "green", "red", "red")

You can create a vector with different functions:

`c()`	function to **c**ombine indvididual values
`seq()`	to create more complex **seq**uences
`rep()`	to create **rep**licates of values

Examples

```
# ============================
# c()function
# ============================
c(2, 7, 8, 12, 3, 25)
# You can assign a vector to a variable
myVector <- c(2, 7, 8, 12, 3, 25)
# You can indicate ranges with the : operator
c(2:5, 3:6)
# You can create a vector with named values
Students <- c(master = "Bob", bachelor = "Lucy", phd = "Jenny")
# ============================
# seq() function
# ============================
seq(from = 1, to = 10)
seq(1,10)
seq(from = 1, to = 10, by = 2)
seq(1, 10, 2)
1:10
# ============================
# rep() function
# ============================
rep(1, 4)
rep(4:6, 3)
rep(1:4, each = 2)
rep(1:4, times = 2, each = 2)
```


Vectors may only have one type.

But – what happens if I ignore this rule?

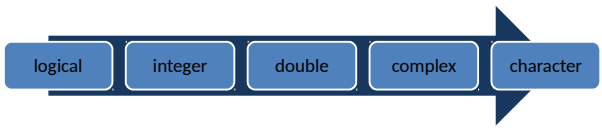
```
x <- c(5, "b")
```

A) The vector is not created and I get an error message.
B) The vector is not created and I get a warning message.
C) The vector is created and I get a warning message.
D) The vector is created without any warning or error.

Data type conversion

- Vectors may only have one type
- When combining different types, R will *coerce* a vector into the most flexible type

Coercion rule (also: *implicit* type conversion)




```
> x <- c(5, "b")
> y <- c(FALSE, 3)
> typeof(x)
[1] "character"
> typeof(y)
[1] "double"
```


Operations on vectors

You assess elements of a vector with the [ ]-operator:

```
x <- c(12, 15, 13, 17, 11)
x[4]
x[3:5]
x[-2]
x[-(3:5)]
```

Standard operations on vectors are element-by-element:

```
c(2, 5, 3) + c(4, 2, 7)
2 + c(2, 5, 3)
c(2, 5, 3) ^ 2
```

Operations on vectors

```
sum(5:7)
prod(4:6)
x <- 1:5
x[3:5]
x[-2]
x > 3
```

Some useful functions (x is an example vector)

`class(x)`	returns class of vector x
`length(x)`	returns the total number of elements
`x[length(x)]`	returns last value of vector x
`rev(x)`	returns reversed vector
`sort(x)`	returns sorted vector
`unique(x)`	returns vector without multiple elements

Factors

- A factor is used to store 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

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

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

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
```

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

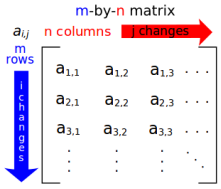
Matrix - Definition

Wikipedia:

In mathematics, a ***matrix*** (plural matrices) is a rectangular array of numbers, symbols, or expressions arranged in ***rows*** and ***columns***. The individual items in a matrix are called its ***elements*** or ***entries***. An example of a matrix with 2 rows and 3 columns is:

$$\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -6 \end{bmatrix}$$

2-by-3 matrix
2x3 matrix



Each element of a matrix is often denoted by a variable with two subscripts. For instance, a2,1 represents the element at the second row and first column of a matrix A.

Matrices - Basics

You can create matrices by:

1. matrix()
2. converting a vector into a matrix
3. binding together vectors

These functions are useful:

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

How does matrix() work?

Function of interest: matrix()

Which arguments can be used with this function?

```
?matrix()
```

matrix {base}

Matrices

Description

`matrix` creates a matrix from the given set of values.

`as.matrix` attempts to turn its argument into a matrix.

`is.matrix` tests if its argument is a (strict) matrix.

Usage

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

```
# Example usage
matrix(data = (1:8), nrow = 2, ncol = 3)
```


A short quiz

Usage

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

Function of interest: `matrix()`

Which version works?

```
matrix(data = (1:6), nrow = 2, ncol = 3)        ✓
matrix(ncol = 3, data = (1:6), nrow = 2)        ✓
matrix(1:6, 2, 3)                               ✓
matrix(1:6, 3, 2)                               ✓
matrix(1:6, ncol = 3, nrow = 2)                 ✓
matrix(1:6, nr = 2, nc = 3)                     ✓
matrix(d = (1:6), nr = 2, nc = 3)               ✘
```

Error in matrix(d = (1:6), nr = 2, nc = 3) :
argument 1 matches multiple formal arguments

Matrices - Examples

```
m <- matrix(data = 1:8, nrow = 4, ncol = 2)
m
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
# indexing is row-by-column
m[2:3, 1:2]
     [,1] [,2]
[1,]    2    6
[2,]    3    7
```


```
m[3, 2]
m[2, ]
m[ ,2]
m[2:3, 1:2]
```

Matrices – Examples

```
z <- as.matrix(1:6)
z
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```


Matrices – Examples

Examples:

```
cbind(1:3,5:7)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
rbind(1:3,5:7,10:12)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    5    6    7
[3,]   10   11   12
```

More in the exercises!

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

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 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
Remember: 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

```
students <- data.frame(id = 1:3, name = c("Linda", "Laura",
"Laura"))
str(students)      # str() displays internal structure of an R
                   # object
'data.frame':  3 obs. of  2 variables:
 $ id  : int  1 2 3
 $ name: Factor w/ 2 levels "Laura","Linda": 2 1 1
```

Argument stringsAsFactors = FALSE prevents this behaviour

Data frames

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

```
students <- data.frame(id = 1:3, name = c("Linda", "Laura", "Laura"),
stringsAsFactors = FALSE)
str(students)
'data.frame': 3 obs. of  2 variables:
 $ id  : int  1 2 3
 $ name: chr  "Linda" "Laura" "Laura"
```

But – what should I do when some variables ARE factors?

```
students <- data.frame(id = 1:3, name = c("Linda", "Laura", "Laura"),
program = c("Master", "Phd", "Phd"), stringsAsFactors = FALSE)
str(students)
'data.frame': 3 obs. of  3 variables:
 $ id     : int  1 2 3
 $ name   : chr  "Linda" "Laura" "Laura"
 $ program: chr  "Master" "Phd" "Phd"
```

Solution – take control on what you are doing (in general a good idea)

```
students <- data.frame(id = 1:3, name = c("Linda", "Laura", "Laura"),
program = as.factor(c("Master", "Phd", "Phd")), stringsAsFactors =
FALSE)
```


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 tomorrow!

What you should know after day 3

Part I: Data types and structures in R

- What are types and structures
- Vectors and factors
- Lists
- Matrices
- Data frames

Part II: Accessing data

Indexing

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]                x[1:4]
[1] 16                 [1] 10 11 12 13

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


Indexing

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
```

Indexing

Indexing by integer vector

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

```
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
```


Indexing

Indexing by integer vector

The entire mth 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
```

The entire nth 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
```


Indexing

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

     [,1] [,2]                         [,1] [,2]
[1,]   10   13                    [1,]   10   16
[2,]   11   14                    [2,]   11   17
```


Indexing

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
```


Indexing

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

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 data structures in R

	single type	multiple types
1D	vector	list
2D	matrix	data frame

Take-home message

- R has different data types and data structures
- Access elements with [ ] or $