Fachhochschule Südwestfalen
University of Applied Sciences

Programmierung R - Exercise

Functions and data types

April 5, SS 2022 || Hannah Behrens

Wir geben Impulse

Task 1

Your turn

Take a look at the function `sd()` from the R package `stats`.

1.1 Which values does this function take as input, i.e. what are its arguments?

1.2 How can you get the function's arguments?

Task 1 - answer

Take a look at the function `sd()` from the R package `stats`.

1.1 Which values does this function take as input, i.e. what are its arguments?

```
args(sd) # get the function's arguments
```

```
## function (x, na.rm = FALSE)
## NULL
```

```
?sd # get detailed information
```

`sd()` consists of the arguments

- `x`: a numeric vector and
- `na.rm`: logical value (TRUE or FALSE) if missing values should be removed

1.2 How can you get the function's arguments?

```
args(...) # apply the function args() to a function to get the arguments of the latter

?functionName # get detailed information about a function
```

Task 2

Your turn

2.1 How can a function be easily constructed in R?

2.2 Why do we need functions in R?

2.3 Write a function that takes an arbitrary value as input, converts the input to a logical value and returns this logical value. Test your function by applying it to the single values -3, 0, 1 and 10! What do you conclude from the results?

Tasks 2.1 and 2.2 - answers

2.1 How can a function be easily constructed in R?

```
functionName <- function(argument1, argument2, ...){
  # function body begins with "{" and ends with "}"
  #...
  # What to do (with the (optional) arguments)
  #...
  return(results) ## return the result (a value, a matrix or a list)
}
```

2.2 Why do we need functions in R?

Functions are like *containers* which fulfill (special) tasks that will be executed several times.

Imagine you have to execute some (special) calculations over and over again but with different start values and parameters. In order to avoid repeating code in your programming project, i.e. avoid copy and paste of code, it is useful to create a function that can be called several times.

Notes

Task 2.3 - answers

2.3 Write a function that takes an arbitrary value as input, converts the input to a logical value and returns this logical value. Test your function by applying it to the single values -3, 0, 1 and 10! What do you conclude from the results?

```
convertToLogical <- function(x){
  logical_value <- as.logical(x) # conversion to logical value
  return(logical_value) # return logical value
}
convertToLogical(x = -3)
```

```
## [1] TRUE
```

```
convertToLogical(x = 0)
```

```
## [1] FALSE
```

```
convertToLogical(x = 1)
```

```
## [1] TRUE
```

```
convertToLogical(x = 10)
```

```
## [1] TRUE
```

Task 2.3 - answers

2.3 Write a function that takes an arbitrary value as input, converts the input to a logical value and returns this logical value. Test your function by applying it to the single values -3, 0, 1 and 10! What do you conclude from the results?

```
convertToLogical <- function(x){
  logical_value <- as.logical(x) # conversion to logical value
  return(logical_value) # return logical value
}
convertToLogical(x = -3)
```

```
## [1] TRUE
```

```
convertToLogical(x = 0)
```

```
## [1] FALSE
```

```
convertToLogical(x = 1)
```

```
## [1] TRUE
```

```
convertToLogical(x = 10)
```

```
## [1] TRUE
```

Conclusion:

if $x \neq 0$: TRUE

if $x = 0$: FALSE

Task 3

Your turn

Expand your function from Task 2.3 by

- allowing three values as input and
- returning the sum of the three input values after converting the input values to logical values.
- What do you expect, when you apply your function to the values -3, 1 and 0?

Task 3 - answer

Two possible solutions:
avoid first solution and prefer second solution!

```r
## first solution:
convertToLogical1 <- function(x1, x2, x3){
  logic_val1 <- as.logical(x1)
  logic_val2 <- as.logical(x2)
  logic_val3 <- as.logical(x3)
  sum_logic_val <- logic_val1 + logic_val2 + logic_val3 # sum of the converted values
  return(sum_logic_val)
}

## second solution:
convertToLogical2 <- function(x){ # x can be a single value or a vector
  logical_value <- as.logical(x)
  sum_logic_val <- sum(logical_value)
  return(sum_logic_val)
}

convertToLogical1(x1 = -3, x2 = 1, x3 = 0) # call convert_to_logical1()
```

```
## [1] 2
```

```r
convertToLogical2(x = c(-3, 1, 0)) # call convert_to_logical2()
```

```
## [1] 2
```

Task 3 - answer

The second solution should be preferred since

1. the input `x` can be a single value or an arbitrarily long vector and
2. as a result, `convertToLogical2()` is only one line of code longer than the *original* function `convertToLogical()`!

Task 4

4.1 Write a function that

- can handle a vector as input
- that prints out the input vector and returns the type of the vector.

4.2 What happens when you apply your function (task 4.1) to the following vectors? Give an explanation.

```
x1 <- c(2, 3, 4, 1)

x2 <- c("2", 3, 4, 1)

y1 <- c(TRUE, TRUE, FALSE)

y2 <- c(TRUE, TRUE, FALSE, 0)

z <- c(2, "2", TRUE)
```

Task 4.1 - answer

4.1 Write a function that

- can handle a vector as input
- that prints out the input vector and returns the type of the vector.

```
1  vectorType <- function(x){
2    print(x) # print out input vector x
3    return(typeof(x)) # return type of the input vector
4    # alternative: typ_x <- typeof(x), return(typ_x)
5  }
```

Task 4.2 - answers

4.2 What happens when you apply your function to the following vectors? Give an explanation.

```
x1 <- c(2, 3, 4, 1) # only numbers present --> type: double
vectorType(x = x1)
```

```
## [1] 2 3 4 1
```

```
## [1] "double"
```

```
x2 <- c("2", 3, 4, 1) # a string is present --> type: character
vectorType(x = x2)
```

```
## [1] "2" "3" "4" "1"
```

```
## [1] "character"
```

Task 4.2 - answers

```r
y1 <- c(TRUE, TRUE, FALSE) # only logicals present --> type: logical
vectorType(x = y1)
```

```
## [1]  TRUE  TRUE FALSE
```

```
## [1] "logical"
```

```r
y2 <- c(TRUE, TRUE, FALSE, 0) # a number is present --> type: double
vectorType(x = y2)
```

```
## [1] 1 1 0 0
```

```
## [1] "double"
```

```r
z <- c(2, "2", TRUE) # a number, a string and a logical present --> type: character
vectorType(x = z)
```

```
## [1] "2"    "2"    "TRUE"
```

```
## [1] "character"
```

→ Conversion due to coercion to get uniform data types to apply mathematical operations

Notes

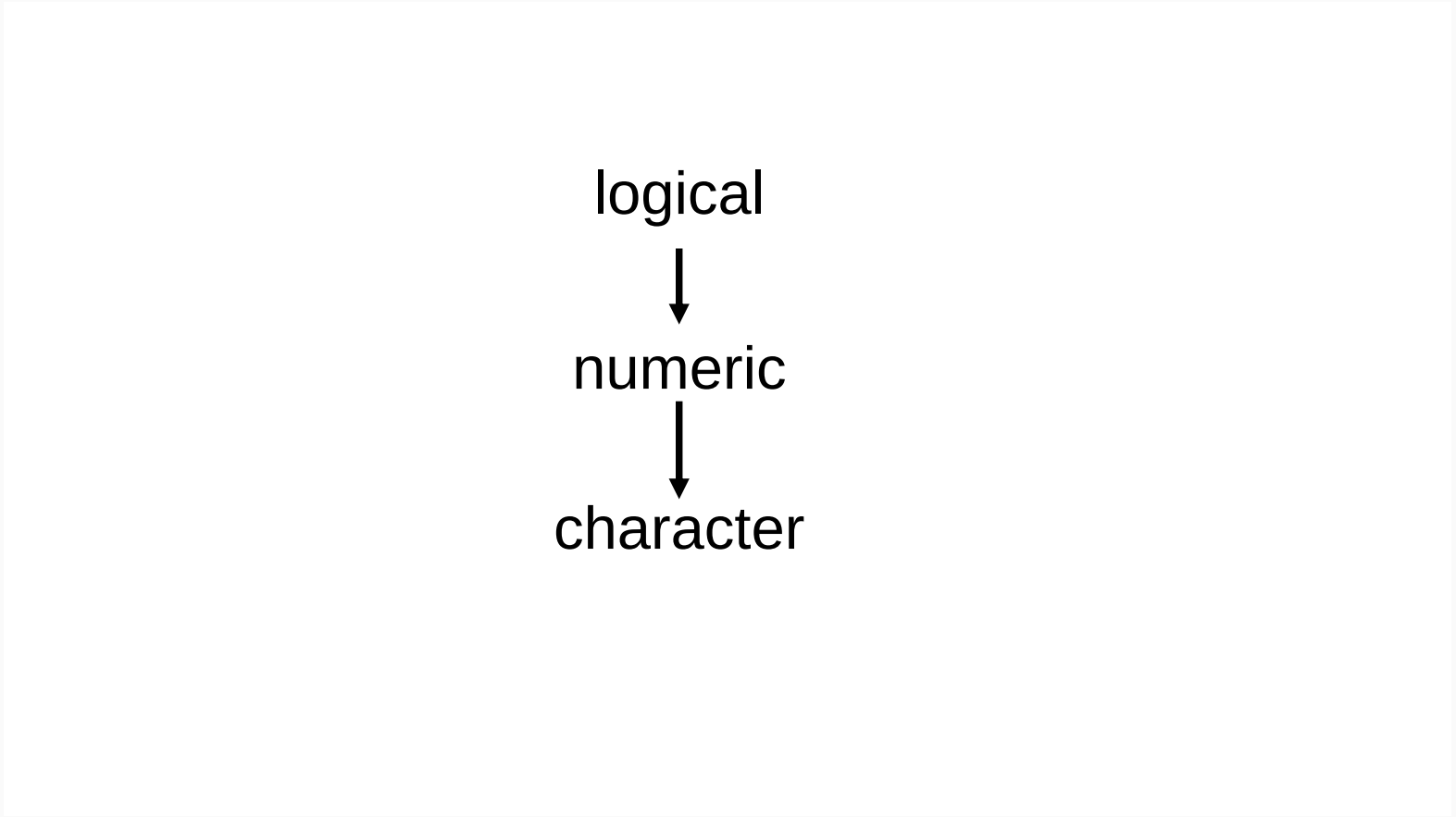

Figure 1: Coercion rules (Buchwitz, 2021).

Task 5

The following vectors are introduced:

```
x <- c(2:8, 21:10, 24)
X <- c(2:8, 10:21, 23)
```

Your turn

Are the following statements correct? Give an explanation and **do not** run the code on your console!

```
# 1)
x %in% X
#>R [1] FALSE

# 2)
X %in% x
#>R [1] TRUE

# 3)
x %in% X == X %in% x
#>R [1] TRUE

# 4)
unique(x %in% X == X %in% x)
#>R [1] TRUE
```

Task 5 - answers

```
# 1) this statement is false since each element of x will be checked whether
# it is part of X; the first 19 elements of x are in X, but not the
# last element (24), so 19 times TRUE and one time FALSE

# 2) this statement is false since each element of X will be checked
# whether it is part of x; the first 19 elements of X are in x, but not
# the last element (23), so 19 times TRUE and one time FALSE

# 3) this statement is false since the elements of the vectors resulting from statements
# 1) and 2) are compared, so a vector with 20 times TRUE results

# 4) this statement is true, since only TRUEs are listed in the vector resulting from 3)
```

Task 6

The following vectors are introduced:

```
y1 <- 13
y2 <- 14
z <- NULL
```

Your turn

Are the following statements correct? Give an explanation and **do not** run the code on your console!

```
# 1)
(y1+y2) >= 27 || is.null(z)
#>R [1] TRUE

# 2)
(y1+y2) >= 27 || !is.null(z)
#>R [1] TRUE

# 3)
(y1+y2) >= 27 || !is.null(w)
#>R [1] TRUE
```

Task 6

```r
# 4)
(y1+y2) >= 27 | !is.null(w)
#>R [1] TRUE

# 5)
!is.null(w) || (y1+y2) >= 27
#>R [1] FALSE
```

Task 6 - answer

```
# 1) this statement is true since the first element ((y1+y2) >= 27) is TRUE

# 2) this statement is true since the first element ((y1+y2) >= 27) is TRUE

# 3) this statement is true since the first element ((y1+y2) >= 27) is TRUE,
# although variable w does not exist

# 4) this statement is false since the variable w does not exist
# consequently, the elements cannot be compared with each other

# 5) this statement is false since the first element, variable w, does not exist
# consequently, the comparison has been stopped
```

Literature - recommendations

Eubank, Randall L., and Ana Kupresanin. 2012. Statistical computing in C++ and R. Boca Raton [u.a.]: CRC Press.

Rizzo, Maria L. 2019. Statistical computing with R. 2nd ed. Boca Raton [u.a.]: CRC Press.

References

Buchwitz, B. 2021. *Computational Statistics*.
`https://bchwtz.github.io/bchwtz-cswr/`.