

R Basics - Functions

Introduction to R - Day 1

Instructor: **Selina Baldauf**

Freie Universität Berlin - Theoretical Ecology

2021-08-01 (updated: 2022-08-30)

Functions

Functions make multiple operations available under one command.

Functions in R:

- Have a **name**
- Have (0, 1, or any number of) **arguments as input**
- They calculate something using the arguments
- They have a **return value** (the output)
- Can have **side effects** (like plotting)

General structure of a function call:

```
function_name ( argument = value )
```



The mean function

`function_name (argument = value)`

`mean (x = c(1,5,6))`

```
mean(x = c(1,5,6)) # or short: mean(c(1,5,6))
```

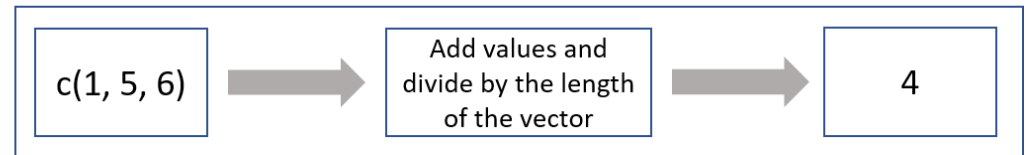
```
## [1] 4
```

- Arguments can also be variables
- The output of a function can be stored in a variable

```
values <- c(1,5,6)
result <- mean(x = values)
result
```

```
## [1] 4
```

mean



The mean function

But what does the mean function do? What are the arguments that I can use?

→ Call the function help using `?`

```
?mean
```

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:
```

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

- x** An `R` object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.
- trim** the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.
- na.rm** a logical value indicating whether `NA` values should be stripped before the computation proceeds.
- ...** further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

Function arguments

- Arguments are the **input** to a function
- Functions can provide **default values** for some arguments
- Default values for arguments are indicated in the function help

```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

<code>x</code>	An <code>R</code> object. Currently there are methods for numeric/logical vectors and date , date-time and time interval objects. Complex vectors are allowed for <code>trim = 0</code> , only.
<code>trim</code>	the fraction (0 to 0.5) of observations to be trimmed from each end of <code>x</code> before the mean is computed. Values of <code>trim</code> outside that range are taken as the nearest endpoint.
<code>na.rm</code>	a logical value indicating whether <code>NA</code> values should be stripped before the computation proceeds.
<code>...</code>	further arguments passed to or from other methods.

Function arguments

```
# NA is a missing value  
values <- c(1, 5, 6, NA)  
mean(x = values)
```

```
## [1] NA
```

What happened? → `na.rm` argument is `FALSE` by default.

Set it to `TRUE` if you want to calculate the mean despite missing values:

```
mean(x = values, na.rm = TRUE)
```

```
## [1] 4
```

Arguments with default values are optional, arguments without default values are not!

```
mean()
```

```
## Error in mean.default(): argument "x" is missing, with no default
```

Function arguments

Argument matching can be achieved by **position** or by **name**

```
mean(x, trim = 0, na.rm = FALSE, ...)  
values <- c(1, 5, 6, NA)
```

These calls to mean all are the same:

```
mean(values, , TRUE) # by position
```

```
mean(x = values, na.rm = TRUE) # by name
```

```
mean(na.rm = TRUE, x = values) # this also works but is not really nice
```

```
mean(values, na.rm = TRUE) # a mix of both
```

Function arguments

Argument matching can be achieved by **position** or by **name**

- Named arguments are (generally) preferred
 - Easier to remember
 - Easier to read
 - Some functions have a lot of arguments

```
mean(x = values, na.rm = TRUE) # by name
```

- However, it is common to match the first argument by position (especially when the first argument is the data)

```
mean(values, na.rm = TRUE) # a mix of both
```


Where do functions come from?

- Built into R (**base R functions**)
- From **additional packages**
 - Packages must be installed first
 - Call a function from a package using `packageName::functionName()`
 - Load the package with `library(packageName)` and then use the function
- **Custom functions**
 - Write your own functions and then use them in the code

```
mean() # calculate mean
seq()  # generate a sequence of values
[]     # This is also a (special) function
```

```
# use read_csv function from readr package
readr::read_csv()

# or use library()
library(readr)
read_csv()
```

```
# custom function that prints input in a
sentence
my_function <- function(x) {
  return(x + 5)
}
my_function(5)
```

Summary

- Functions take **input** in the form of (named) arguments, calculate something and **return** a result
- Functions are called by their name, followed by parentheses:
`functionName (argument1 = value, argument2 = value, ...)`
- Functions from additional packages can be called in two ways:
 - `packageName::functionName()`
 - first load the package with `library(packageName)` then call the function anywhere in the script with `functionName()`
- Call `?functionName` to open the help of a function