Intro to R

Part 2: R Objects

QuantArch Week 1 | 07-02-2022

Universiteit Leiden

Functions and data types in R

Discover the world or something like that…

Functions

Usually containing a series of commands to automate a process.

We have already been introduced to a couple of built-in R functions

- `rm()`
- `print()`

These performed specific **functions** when we gave them an object,

i.e. remove or print the given object.

Anatomy of a function

A function can be called with the function name, and **input arguments** within the brackets:

```
> example_function(arg_1 = some_argument, arg_2 = some_other_argument)
```

This will then provide some sort of output/result (if it worked…)

You can save the output of a function to an object,

just like the calculations from before the break

Anatomy of a function

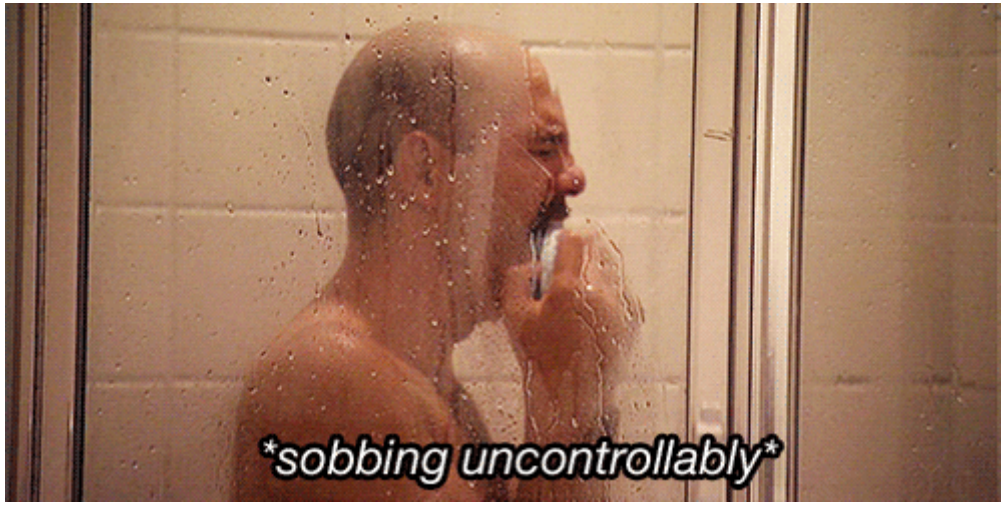
I wouldn't recommend looking inside a function...

```
median
```

```
## function (x, na.rm = FALSE, ...)
## {
##     if (is.factor(x) || is.data.frame(x))
##         stop("need numeric data")
##     if (length(names(x)))
##         names(x) <- NULL
##     if (na.rm)
##         x <- x[!is.na(x)]
##     else if (any(is.na(x)))
##         return(x[FALSE][NA])
##     n <- length(x)
##     if (n == 0L)
##         return(x[FALSE][NA])
##     half <- (n + 1L)%/%2L
##     if (n%%2L == 1L)
##         sort(x, partial = half)[half]
##     else mean(sort(x, partial = half + 0L:1L)[half + 0L:1L])
## }
```

Anatomy of a function

I wouldn't recommend looking inside a function...



Example functions

```
my_numbers <- c(1:10) # start by creating a vector
my_numbers
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
mean(x = my_numbers) # functions will often have obvious names...
```

```
## [1] 5.5
```

```
sum(x = my_numbers) # see what I mean?
```

```
## [1] 55
```

In these functions, the first argument (`x`) is a vector. As long as you put arguments in the correct order, you don't need `x =`

```
mean(my_numbers) # it assumes I mean x = my_numbers
```

```
## [1] 5.5
```

Help

> *Help will always be given in R to those who ask for it.*
>
> Dumbledore (paraphrasing, again…)

To see which arguments a function requires

or to get additional help for a function

just type the name of the function preceded by ?

```
?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.

Basic data types in R

Data type	Explanation
vector/array	a series of values, which can be numeric, logical, or strings
matrix	a table of the same type of values, e.g. all numeric
data frame	a table with different types of values. Can combine strings and numbers
list	a series of data types. Can combine vectors, matrices, and data frames

Data types with examples

Vector

```
c(1, 2, 5.5, TRUE, "random string")
```

```
## [1] "1"             "2"             "5.5"           "TRUE"
## [5] "random string"
```

Matrix 😎

```
matrix(1:12, nrow = 3)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Data types with examples, the sequel

Data frame

```
data.frame("names" = c("Arthur Dent", "Ford Prefect", "Trillian"),
           "occupation" = c("???", "Researcher", "Astrophysicist"),
           "coolness" = c(3, 2, 1))
```

```
##          names     occupation coolness
## 1  Arthur Dent            ???        3
## 2 Ford Prefect     Researcher        2
## 3     Trillian Astrophysicist        1
```

Data types with examples, the sequel, part 2

List

```
# I cheated and stored the objects in advance (i.e. vect, m, datf)
list("vector" = vect, "matrix" = m, "data_frame" = datf)
```

```
## $vector
## [1] "1"             "2"             "5.5"           "TRUE"
## [5] "random string"
##
## $matrix
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## $data_frame
##          names     occupation coolness
## 1  Arthur Dent            ???        3
## 2 Ford Prefect     Researcher        2
## 3     Trillian Astrophysicist        1
```

Exercises

Atomic R object types

typeof()	mode()
logical	logical
integer	numeric
double	numeric
complex	complex
character	character

Here are some examples of predefined operators in R:

`FALSE`, `NA`, `NaN`, `NULL`, `TRUE`

Use the `typeof` and `mode` functions to determine what category these operators belong to.

Exercises

Solution

typeof()	mode()
logical	logical
integer	numeric
double	numeric
complex	complex
character	character

`TRUE`, `FALSE`, `NA`: logical

`NaN`: numeric, double

`NULL`: NULL

Exercises

Create and store a vector with the values `1.5`, `FALSE`, `"universe"`, `NA`.

Use the `length` function to compute the number of elements in the vector, and use `typeof` to see the type of the created object.

Solution

```
my_vector <- c(1.5, FALSE, "universe", NA)

typeof(my_vector)
```

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

A single vector can only contain one atomic object type, so it converted all elements to `"character"`.

character > double > integer > logical

Vectors

We're going to dive into vectors in a little more detail.

There is a difference in vector definitions across the sciences.

In computer science (most relevant here), it is a one-dimensional array that we can use to store values/data.

In R, we create a vector using the `c` function.

Let's create and print a vector with a series of values:

```
artefacts <- c("Sankara Stone", "Ark of the Covenant", "Holy Grail", "Crystal Skull", "Sankara
artefacts
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Holy Grail"
## [4] "Crystal Skull"       "Sankara Stone"
```

This will of course be a character vector with five elements.

```
typeof(artefacts)
```

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

```
length(artefacts)
```

```
## [1] 5
```

We can also create a logical vector:

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

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

```
typeof(boolean)
```

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

```
length(boolean)
```

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

This is also known as a 'Boolean' vector (and is great for indexing).

What happens if we convert this to a numeric vector and a character vector?

```
as.numeric(boolean)
```

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

```
as.character(boolean)
```

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

Note: The predefined operators (e.g. `FALSE`, `NaN`, etc.) lose their special properties when converted to a character.

Exercises

What type of object do you think the following vectors will be?
Use the `typeof` or `class` function to check.

```
vector_1 <- c(1, 3, "f", 2)
vector_2 <- c(4, TRUE, 5, 1)
vector_3 <- c("life", "universe", "everything", FALSE)
vector_4 <- c(3, 1, 4, 1, "5")
```

Reminder: Vectors can only hold one data type. Priority for vector conversion:

character > double > integer > logical

Solution

```
class(vector_1)
```

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

```
class(vector_2)
```

```
## [1] "numeric"
```

```
class(vector_3)
```

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

```
class(vector_4)
```

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

Exercises

What happens if we combine `vector_2` and `vector_3`?
Hint: The `c` function can also combine vectors.

```
vector_1 <- c(1, 3, "f", 2)
vector_2 <- c(4, TRUE, 5, 1)
vector_3 <- c("life", "universe", "everything", FALSE)
vector_4 <- c(3, 1, 4, 1, "5")
```

Solution

```
c(vector_2, vector_3)
```

```
## [1] "4"          "1"          "5"          "1"          "life"
## [6] "universe"   "everything" "FALSE"
```

What happened to the `TRUE` entry in `vector_2`? 🤷

Missing values

Missing values in R, are represented with NA.

```
artefacts <- c(artefacts, NA) # add NA to the end of our artefacts vector
artefacts
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Holy Grail"
## [4] "Crystal Skull"       "Sankara Stone"       NA
```

Keep this in mind when you are collecting data.

As you can see, NA is allowed to keep its special property in the various vector types.

```
is.na(artefacts)
```

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

Working with the data types

Subsetting with vectors

Let's look at the `artefacts` again. We can extract elements of the vector by subsetting, square brackets `[]`, in various ways.

We can do it by position using vectors:

```
artefacts[3] # extract the third element
```

```
## [1] "Holy Grail"
```

```
artefacts[c(1,4)] # extract first and fourth elements
```

```
## [1] "Sankara Stone" "Crystal Skull"
```

```
artefacts[-3] # extract everything EXCEPT the third element
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Crystal Skull"
## [4] "Sankara Stone"       NA
```

Subsetting with conditions

Or we can do it with conditional vectors:

```
artefacts[c(TRUE, TRUE, TRUE, FALSE, TRUE)] # drop the fourth element
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Holy Grail"
## [4] "Sankara Stone"       NA
```

Note: The conditional vector must be the same length as the vector you are subsetting. If not, it will recycle elements of the conditional vector

And conditional statements:

```
artefacts[artefacts == "Sankara Stone"] ## NOTE the double '=='
```

```
## [1] "Sankara Stone" "Sankara Stone" NA
```

```
# The statement actually just generates a conditional vector
artefacts == "Sankara Stone"
```

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

Subsetting with conditions

If we have a numeric vector, we can subset with numeric conditions:

```
my_numeric <- runif(6, 0, 10) # 6 random numbers between 0 and 10
my_numeric[my_numeric < 5] # extract numbers less than 5
```

```
## [1] 3.1834130 0.8938124 0.2919744
```

And you can use multiple conditional statements with & (and), or | (or):

```
my_numeric[my_numeric > 2 & my_numeric < 8] # numbers between 2 and 8
```

```
## [1] 7.485215 6.763866 3.183413
```

Exercises

Subset everything except the 3rd and 4th values from the `artefacts` vector.

Solution

```
artefacts[-c(3, 4)] # most efficient solution
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Sankara Stone"
## [4] NA
```

```
artefacts[c(TRUE, TRUE, FALSE, FALSE, TRUE, TRUE)]
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Sankara Stone"
## [4] NA
```

```
artefacts[c(1,2,5,6)]
```

```
## [1] "Sankara Stone"       "Ark of the Covenant" "Sankara Stone"
## [4] NA
```

Modifying vectors

We can use indexing to replace values in a vector

```
artefacts[2] <- NA # replace the second element with NA
is.na(artefacts)
```

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

Vector operations

We can perform operations on a vector using the operators listed earlier.

```
my_numeric * 3 # multiply all elements by 3
```

```
## [1] 24.5684346 22.4556450 20.2915993  9.5502391  2.6814373  0.8759232
```

```
my_numeric / 3 # divide all elements by 3
```

```
## [1] 2.7298261 2.4950717 2.2546221 1.0611377 0.2979375 0.0973248
```

We can also use functions on a vector:

```
mean(my_numeric) # find the average of the vector
```

```
## [1] 4.46796
```

```
round(my_numeric) # round all elements to a single digit (see ?round)
```

```
## [1] 8 7 7 3 1 0
```

Done for the day

When closing your RStudio session, you may be prompted to save workspace data. Do not save. It is best to start with a fresh session when you are working on an R Script.

You can disable automatically loading .RData into RStudio by navigating to

Tools > Global Options (or Project Options) > General

Then untick the .RData box under the **Workspace** header.

Common errors

`Error in <function-name> : could not find function "<function-name>"`

- Typo
- Using a non-function object as a function (Error: attempt to apply non-function)

`Error: object '<name>' not found`

- Typo
- Did you remember to store the object?

💡 **Tip**

Your preferred search-engine is a very useful helper.