

Tutorial - Part 1

Marine Ecosystem Dynamics - 2025

R syntax

R is a programming language that use a simplified syntax. In this section, we will explore how to write a script and execute it.

But first some syntax information:

- Everything after # is considered as a comment and will not be executed. It is very important to write what we are doing, so we do not get lost next time we open our scripts.

```
# 2 + 2 will not work because of the #
2 + 2 # We should then annotate our script like this
#> [1] 4
```

- Several lines of code can be written in one line but must be separated by a semicolon

```
2 + 2
#> [1] 4
3 * 2
#> [1] 6

# This can also be written as follow:
2 + 2 ; 3 * 2
#> [1] 4
#> [1] 6
```

- In **R** we can name any object using =, <-, -> or assign

```
c(1, 2, 3, 4) -> my_first_vector
my_vector <- c(1, 2, 3, 4)
my_function = function(x){x + 2}
assign("x", c(2, 3, 4, 5))
```

- == is a logical function that can be translated as *is equal to*, contrarily *is not equal to* is written !=

```
2 + 2 == 4
#> [1] TRUE
3 * 2 == 4
#> [1] FALSE
3 * 2 != 4
#> [1] TRUE
```

Exercises

Using a new **R** script, do these calculations:

- 2^7
- $\cos(\pi)$
- The sum of all number from 1 to 100

Create a parameter `x1` that equals to 5 and a parameter `x2` that equals to 10

- Is $2 * x1$ equal to `x2`?

Functions

As seen during the lecture, **R** works with functions that can:

- Already be implemented in base **R**
- Coming from another package
- Created by the user

We will see these three examples in this section, but first it is important to remember that the typical structure of a function is `function(argument1, ...)`.

Fortunately **R** helps us to remember what are the needed arguments:

- Using `help()` or `?`
- Using `example`

For the functions that comes from external packages, we first need to install the new packages. The most common way to do so is by executing `install.packages("Package_Name")`. Then when we want to load the functions, we start the script by executing `library(Package_Name)`.

Finally, if we really do not find a suitable function in a package, we can create your functions following this general structure, but this will not be covered in this tutorial:

```
my_function <- function(<argument1>, <argument2>, ...){  
  <here comes the definition of my function>  
  return(<output of the definition>)  
}
```

Exercises

- What is the function `log()` doing and from where does this function come from (base **R**, other packages)?
- What are the mandatory arguments for the function `plot()`
- Is there help associated with the functions from a loaded package?

Vectors

R works with vector from which we can do our calculations.
Several ways exist to create a vector:

- Using `c()`, values are added next to each other and separated with a `,`.

```
c(1, 2, 1, 4) # It works with integers (round numbers)
c(1.1, 2.4, 3.14652) # It works with floats (decimal numbers)
c("chocolate", "ice-cream") # It works with character
c(TRUE, FALSE) # It works with logical variables
```

- Using `rep()` to repeat the same values several times.

```
rep(x = 3, 2) # it reads: repeat 2 times the value x that is equal to 3
rep(x = "chocolate", 3)
# it reads: repeat 3 times the value x that is equal to "chocolate"
```

- Using `seq()` to create a sequence of values. It only works for numeric values!

```
seq(from = 0, to = 10, by = 2) # it reads: create a sequence of values from 0
# to 10 every 2 numbers
seq(from = -1, to = 1, by = 0.2) # it also works with negative values and
# decimal
```

- Combining all of the above

```
rep(x = c(seq(from = 2, to = 3, by = 0.2), 5), 2)
c(rep(x = "character", 5), "other character")
c(seq(from = 2, to = 10, by = 2), rep(x = 1000, 2), c(1, 4, 2))
```

Exercises

- Create a vector `v1` that contains the values 1, 2, 3, 4, 6
- Create a vector `v2` that contains 10 times the values 1, 2, 3, 4, 6
- Create a vector `v3` that repeats `TRUE`, `FALSE` 2 times
- Create a vector `v4` that goes from 10 to 2000
- Create a vector `v5` that contains `v1`, `v2`, `v3` and 2 times `v4`

Dataframe

Most likely, we will work with data stored in dataframes. A dataframe is composed of observations (rows) and variables (columns). We can see a dataframe like multiples vectors put together.

For example in the dataframe below (named `df`) is composed of 4 vectors:

1. Species that contains the species names
2. Abundance that contains the abundances of the species
3. Location that contains the location of the species
4. Date that contains the sampling date

```
#>      Species Abundance Location      Date
#> 1      Acartia         34     Askö 03-09-2024
#> 2 Pseudocalanus         12     Askö 04-09-2024
#> 3   Centropages         17     Askö 02-09-2024
```

We can access the individual columns (i.e., vectors) using `$`

```
df$Species
#> [1] "Acartia"      "Pseudocalanus" "Centropages"
df$Abundance
#> [1] 34 12 17
df$Location
#> [1] "Askö" "Askö" "Askö"
df$Date
#> [1] "03-09-2024" "04-09-2024" "02-09-2024"
```

Exercises

- Create a vector `genus` containing the character "Acartia", "Centropages", "Temora", "Acartia", "Centropages", "Temora"
- Create a vector `station` containing the character "Askö", "Askö", "Askö", "Tjarnö", "Tjarnö", "Tjarnö"
- Create a vector `abundance` containing the values 3, 10.2, 4, 2.3, 4, 9.4
- Combine all the vectors in a dataframe called `df`
- Create a vector output that correspond to the column `Abundance` of the dataframe `df`. Is output similar to the vector `abundance`?

Importing data in **R**

More often we enter our data in spreadsheets. We then need to import our data in **R** to process them.

To do so, we use the `read.*` function family.

A typical data import protocol looks like this:

1. Set the working directory with its absolute path

```
setwd("/Absolute/Path/To/Working/Directory")
```

1. Import your dataset in your environment

```
df <- read.csv("./Relative/Path/Dataset.csv")
```

1. Examine the structure of the data to see if the importation worked well

```
str(df)  
head(df)  
tail(df)
```

Exercises

You can download the dataset on [GitHub](#)

- Import the dataset in your environment
- How many rows and columns does this dataset contain?
- What are the headers of the columns?
- What is the last row?