# Methods in evolutionary ecology WS25/26

Stefanos Siozios Michael Gerth

# Table of contents

# About

This script covers the computational and bioinformatics parts of the module "Methods in Evolutionary Ecology". We will introduce you to `R` and `BASH`, two of the most widely used scripting languages, and make you familiar with navigating in a UNIX environment. These skills are important for any biologist, irrespective of the field you may want to specialise in in the future. Building upon your new knowledge, we will learn how to reconstruct phylogenies from sequencing data, how to work with genomic data, and how to characterise microbiomes. At the end of three weeks computational work, you will tackle a small computational group project, putting your new skills into practise.

The script is designed to cover the entire course content. While we will go you through all of the material together in detail during the course, the script should also enable you to work through the content on your own, e.g., to recap after the course has finished and as a reference and starting point for future computational endeavours.

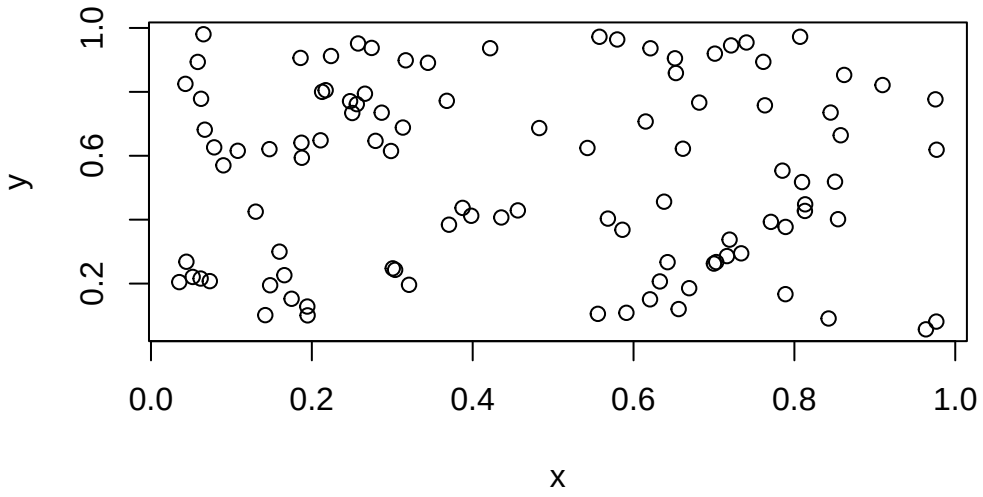
## Quarto

The text is formatted using Quarto, which comes with a number of benefits. It allows us to provide explanations as structured and nicely formatted regular text, and to include code blocks for all computational steps. When compiling Quarto documents, all of the code is run, which means that you not only see the code, but also the outputs it creates.

Here is an example:

This little block of `R` code generates 100 random coordinates and plots them. The code is shown below, together with the output the code has produced (in this case, a plot).

```
x <- runif(100)
y <- runif(100)
plot(x, y)
```




The code can conveniently be copied from the block into your own scripts.

Quarto supports many formats, we here provide the script as a webpage and a printable pdf. Writing Quarto documents is very simple and can be done using RStudio as an editor. The entire script is available for you on github – feel free to download it and modify it with your own comments, notes, and code. We will provide a short introduction to github and Quarto in the course.

## How to find your way around

Simply use the navigation on the left to quickly access the different topics, or flip through the individual pages using the buttons at the bottom of the page. You may wish to download the pdf version of the script (click the pdf icon in the top left) which is ideal for printing. The script is organized by topics, rather than course days, because we will adapt the tempo according to your needs.

> Please note: The script will very likely only be complete at the end of the course. We will still be modifying and correcting it throughtout the three weeks you are with us. So make sure to check out the final version at the end of the course.

# Part I

# R

# 1 First steps

R is a statistical programming environment that has become a standard tool in the data and life sciences and many other fields. You may have used R already to run some statistics in a course you took in your studies, and this will be a likely use case for your remaining degree. However, R is much more: it can be used to analyse massive the datasets of the "omics"- age, build webpages, blogs, and interactive apps, and even for art!

Before taking full advantage of what the various R packages have to offer, we need to become familiar with its basic structure and commands. It pays off to invest a little effort in practicing the basics, because all R packages use the same syntax – a solid familiarity with base R thus allows you to explore the entire R universe independently.

## 1.1 Operators and functions

R can be used just like an arithmetic calculator. You are familiar with all of the basic syntax already, if you know how to use a calculator!

Some examples:

```
3 + 4
```

```
[1] 7
```

```
3 - 4
```

```
[1] -1
```

```
3 * 4
```

```
[1] 12
```

```
3 / 4
```

```
[1] 0.75
```

```
3 ^ 4
```

```
[1] 81
```

As with a regular calculator, there is operator precedence: power > multiplicative operations > additive operations:

```
(1 + 2) * 3
```

```
[1] 9
```

```
2^3 * 3
```

```
[1] 24
```

```
2^(3 * 3)
```

```
[1] 512
```

Square roots, exponentials, and logarithms also work just as with a calculator:

```
sqrt(9)
```

```
[1] 3
```

```
exp(3)
```

```
[1] 20.08554
```

```
log(3)
```

```
[1] 1.098612
```

```
log(exp(3)) # natural logarithm
```

```
[1] 3
```

```
log10(100)  # logarithm to base 10
```

```
[1] 2
```

In order to "save" a value for use later on, you have to assign it to a variable! `<-` is the assignment operator you need to use for this (handy shortcut in RStudio is `ALT` + `-`).

```
x <- 3 + 4
```

Calling the variable will then print the result to the R console, and can be used in other calculations.

```
x
```

```
[1] 7
```

```
x + 10
```

```
[1] 17
```

You can call your variables whatever you want, but be careful: `R` will overwrite any variable if you tell it to, without a warning! You should also avoid giving your variables names that are already assigned to functions.

```
my_favourite_variable <- 100
my_favourite_variable <- 50
my_favourite_variable
```

```
[1] 50
```

All variables (among other things) are visible in the environment panel in RStudio (default: top right part of the screen).

> "=" can also be used to assign variables but is discouraged, because the direction of the assignment is not immediately obvious. It is best practise to always start with the variable, followed by the assignment operator

## 1.2 Data types

You need to be familiar with at least three important data types in `R`: `logical`, `numeric`, and `character`. Data being stored in a different data type than required is one of the most frequent error messages you will encounter as an `R` beginner.

`logical` simply means true or false. `R` also understands the abbreviations T and F. To determine which types you data is in, you can use `mode` or `class`.

```
var1 <- TRUE
mode(var1)
```

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

`numeric` means numbers

```
var2 <- 10
class(10)
```

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

A character is any form of text, a so called "string". It must always be surrounded by quotation marks!

```
var3 <- "A so called string"
mode(var3)
```

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

If in doubt, `R` will often convert or read in data as characters. Watch out for some common errors!

```
var4 <- "5"
var4
```

```
[1] "5"
```

```
is.numeric(var4)
```

```
[1] FALSE
```

```r
var5 <- "TRUE"
var5
```

```
[1] "TRUE"
```

```r
is.logical(var5)
```

```
[1] FALSE
```

You can convert between types easily!

```r
var6 <- as.numeric(var4)
var6
```

```
[1] 5
```

```r
class(var6)
```

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

## 1.3 Exercises

a) Sum the values of 1 to 5

b) Create a variable v1 and assign it a character value

c) Copy variable v1 to v2

d) Compare the value of v1 against v2

> **Tip**
>
> Compare values and variables using the following operators
>
> | | |
> |---|---|
> | < | less than |
> | <= | less than or equal to |
> | > | greater than |
> | >= | greater than or equal to |

| | |
|---|---|
| `==` | equals |
| `!=` | not equal |

Please note, `=` and `==` do very different things! Don't mix them up.

# 2 Data structures

So far we've only looked at simple variables consisting of a single value or character. Typically, your data will be more complex. In R, there are three structures relevant for the data you will be working with.

## 2.1 Vectors

A **vector** is a number of elements of the same data type (`logical`, `numeric`, `character`). It can be generated by concatenating the elements using the function `c`.

```
vec1 <- c(T, F, T, F)
vec1
```

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

```
mode(vec1)
```

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

```
vec2 <- c(1, 2, 3, 4, 5)
vec2
```

```
[1] 1 2 3 4 5
```

```
mode(vec2)
```

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

```
vec3 <- c("Spring", "Summer", "Autumn", "Winter")
vec3
```

```
[1] "Spring" "Summer" "Autumn" "Winter"
```

```
mode(vec3)
```

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

Other ways to generate vectors are `rep` and `seq`. `rep` is used to repeat any number of elements any number of times.

```
rep(5, 10)
```

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

```
rep(vec3, 5)
```

```
 [1] "Spring" "Summer" "Autumn" "Winter" "Spring" "Summer" "Autumn" "Winter"
 [9] "Spring" "Summer" "Autumn" "Winter" "Spring" "Summer" "Autumn" "Winter"
[17] "Spring" "Summer" "Autumn" "Winter"
```

`seq` can be used to create numerical sequences.

```
seq(from = 0, to = 100, by = 5)
```

```
 [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
[20]  95 100
```

The command above is easy to read and understand for humans, which is good. `R` will also understand if you specify it as

```
seq(0, 100, 5)
```

```
 [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
[20]  95 100
```

As a shortcut for a common sequences, you can use

```
1:10
```

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

As mentioned above,, vectors can only combine elements of a single data type. Combining multiple different data types may result in some unwanted behaviour.

```
vec_mix1 <- c(5, TRUE, 65)
mode(vec_mix1)
```

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

```
vec_mix2 <- c("blue", TRUE, "red")
mode(vec_mix2)
```

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

In many cases you may wish to access a single element of a vector. You can do so using square brackets.

```
z <- c("order", "family", "genus", "species")
z[2]
```

```
[1] "family"
```

Similarly, you can access any combination of elements from the vector.

```
z[1:2]
```

```
[1] "order"  "family"
```

```
i <- c(1, 3)
z[i]
```

```
[1] "order" "genus"
```

```
z[c(1, 1, 1, 4)]
```

```
[1] "order"   "order"   "order"   "species"
```

```
z[-1]
```

```
[1] "family"  "genus"   "species"
```

The square brackets are also used if you need to change elements of the vector. Changes are made using the assignment operator which you already know.

```r
x <- 1:5
x
```

```
[1] 1 2 3 4 5
```

```r
x[c(1, 4)] <- 10
x
```

```
[1] 10  2  3 10  5
```

Which elements of a vector have certain characteristics? This is important for filtering/selecting in your dataset. You can combine different queries using logical operators.

```r
x >= 5
```

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

```r
x[x >= 5]
```

```
[1] 10 10  5
```

```r
which(x >= 5)
```

```
[1] 1 4 5
```

```r
z
```

```
[1] "order"   "family"  "genus"   "species"
```

```r
which(z == "genus")
```

```
[1] 3
```

```r
z[z== "genus"]
```

```
[1] "genus"
```

```
z[z != "genus"]
```

```
[1] "order"   "family"  "species"
```

```
which(z== "genus" | z == "order")
```

```
[1] 1 3
```

Logical operators in R

| | |
|---|---|
| \| | OR |
| & | AND |
| ! | NOT |

Conveniently, the elements of a vector can be named and accessed using the names. Let's first create a vector…

```
dmel <- c("Hexapoda", "Diptera", "Drosophilidae", "Drosophila", "Drosophila melanogaster")
dmel
```

```
[1] "Hexapoda"                "Diptera"
[3] "Drosophilidae"           "Drosophila"
[5] "Drosophila melanogaster"
```

… and then add names for each element

```
names(dmel) <- c("Class", "Order", "Family", "Genus", "Species")
dmel
```

```
                Class                 Order                Family
          "Hexapoda"             "Diptera"       "Drosophilidae"
                Genus               Species
          "Drosophila" "Drosophila melanogaster"
```

```
str(dmel)
```

```
 Named chr [1:5] "Hexapoda" "Diptera" "Drosophilidae" "Drosophila" ...
 - attr(*, "names")= chr [1:5] "Class" "Order" "Family" "Genus" ...
```

Now we can use the names to access the values

```
dmel[c("Class", "Species")]
```

```
                Class                   Species
"Hexapoda" "Drosophila melanogaster"
```

```
dmel[names(dmel) == "Order"]
```

```
    Order
"Diptera"
```

### 2.1.1 Exercises

a) Create a vector consecutively numbering all days of the year 2026. Assign the correct weekday names for all elements of the vector.

b) Use the vector to determine how many days in 2026 are weekend days.

> **Tip**
>
> If you struggle to assign the correct names, have a look at the help for `rep`.

## 2.2 Matrices

A **matrix** in `R` can be thought of as a two-dimensional vector. All elements must be of the same data type. There are various ways to create a matrix. For example, one can use the `matrix` function like this.

```
mat1 <- matrix(data = 1:12, nrow = 3, ncol = 4, byrow=T)
mat1
```

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

Alternatively, a vector can be transformed into a matrix

```
mat2 <- 1:12
dim(mat2) <- c(3, 4)
mat2
```

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

Often you will want to combine multiple vectors into a matrix

```
dmel <- c("Hexapoda", "Diptera", "Drosophilidae", "Drosophila", "Drosophila melanogaster")
dhyd <- c("Hexapoda", "Diptera", "Drosophilidae", "Drosophila", "Drosophila hydei")
mat3 <- cbind(dmel, dhyd)
mat3
```

```
     dmel                      dhyd
[1,] "Hexapoda"                "Hexapoda"
[2,] "Diptera"                 "Diptera"
[3,] "Drosophilidae"           "Drosophilidae"
[4,] "Drosophila"              "Drosophila"
[5,] "Drosophila melanogaster" "Drosophila hydei"
```

```
mat4 <- rbind(dmel, dhyd)
mat4
```

```
     [,1]       [,2]      [,3]            [,4]
dmel "Hexapoda" "Diptera" "Drosophilidae" "Drosophila"
dhyd "Hexapoda" "Diptera" "Drosophilidae" "Drosophila"
     [,5]
dmel "Drosophila melanogaster"
dhyd "Drosophila hydei"
```

Just like vectors, matrix elements can have names

```
mat3
```

```
     dmel                       dhyd
[1,] "Hexapoda"                 "Hexapoda"
[2,] "Diptera"                  "Diptera"
[3,] "Drosophilidae"            "Drosophilidae"
[4,] "Drosophila"               "Drosophila"
[5,] "Drosophila melanogaster" "Drosophila hydei"
```

```
colnames(mat3)
```

```
[1] "dmel" "dhyd"
```

```
rownames(mat3) <- c("Class", "Order", "Family", "Genus", "Species")
mat3
```

```
        dmel                       dhyd
Class   "Hexapoda"                 "Hexapoda"
Order   "Diptera"                  "Diptera"
Family  "Drosophilidae"            "Drosophilidae"
Genus   "Drosophila"               "Drosophila"
Species "Drosophila melanogaster" "Drosophila hydei"
```

And just like with vectors, we can use square brackets to access and replace values. Because there are 2 dimensions, we need to provide 2 values (one for rows, one for columns, separated by ,).

```
mat3
```

```
        dmel                       dhyd
Class   "Hexapoda"                 "Hexapoda"
Order   "Diptera"                  "Diptera"
Family  "Drosophilidae"            "Drosophilidae"
Genus   "Drosophila"               "Drosophila"
Species "Drosophila melanogaster" "Drosophila hydei"
```

```
mat3[1:3, 2]
```

```
          Class           Order          Family
     "Hexapoda"       "Diptera" "Drosophilidae"
```

```
mat3[1:3, ]
```

```
       dmel            dhyd
Class  "Hexapoda"      "Hexapoda"
Order  "Diptera"       "Diptera"
Family "Drosophilidae" "Drosophilidae"
```

```
mat3[c("Class", "Species"), ]
```

```
        dmel                       dhyd
Class   "Hexapoda"                 "Hexapoda"
Species "Drosophila melanogaster" "Drosophila hydei"
```

### 2.2.1 Exercise

a) create a matrix using with 20 rows & 5 columns, using 100 randomly generated numbers between 0 and 1000.

b) replace all values in the 3rd column of this matrix that are larger than 500 with `NA`.

## 2.3 Data frames

# 3 Tidyverse

Some stuff about tidyverse

```
1 + 1
```

```
[1] 2
```

# Part II

# UNIX

# Part III

# Phylogenetics

# References