R Intro

Jesús N. Pinto-Ledezma and Jeannine Cavender-Bares

The main goal of this tutorial is to present basic aspects for anyone to be free about initial fear and start using R to perform data analysis. Every learning process becomes more effective when theory is combined with practice, in this sense, we strongly recommend that you follow the exercises in this short tutorial at the same time that you run the commands on your computer, and not just read it passively.

Why R?

R is a language and a statistical programming environment and graphics or also called an **"object oriented programming"**, which means, that using R involves the creation and manipulation of objects on a screen, where the user has to say exactly what they want to do rather than simply press a button **(black box paradox)**. So, the main advantage of R is that the user has control over what is happening and also the fully understanding of what he/she wants before performing any analysis.

With R it is possible to manipulate and analyze data, make graphics and write from small commands to entire programs. Basically, R is the open version of the S language, created by Bell's Lab in 1980. Interestingly, S language is super popular among different areas of sciences and is the base for commercial products such as, SPSS, STATA, SAS among others. Thus, if we have to add another advantage to R, is that R is an **open language and free**!

There are different sources and web-pages with a lot of information about R, most of them are super useful and can be found at DataCamp (https://www.datacamp.com/), CRAN (https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf), R Tutorial (http://www.r-tutor.com/r-introduction).

Also, when we are reporting our results in the form of a report, scientific paper or any kind of document, we would need to cite the used software, the easiest to cite R is using the internal function **citation()**.

First steps

First that all, we need to know about **WHERE** are we working at. In other words, our working directory. To get information that information we just need to type **getwd()** in the script or the console.

```
getwd()
```

```
## [1] "/Users/jesusn.pinto-ledezma/Documents/GitHub/BiodiversityScience/Spring2021"
```

If the working directory is not the correct one, we just need to order R to **SET** the correct address.

There is an R package called **{here}** that is super convenient for setting your working directory if the path is really long. You didn't hear that from me ;p!

Ok, we are now in the correct place, so we can continue with the practice.

Directory structure

For training purposes, we will create a **directory structure** where the main folder is our current working place, so we will create a series of **subfolders** where we store, the data, the scripts and whatever we want... To do that we will use the function **dir.create()**. Let's practice!

```
dir.create("Main-BioSci") # this can be your main folder and you can change the name
```

```
dir.create("Data") # folder that store the data
```

```
## Warning in dir.create("Data"): 'Data' already exists
```

```
dir.create("R-scripts") # folder that store the scripts used in the course
```

```
dir.create("Figures") # folder that store the figures created in the course
```

```
dir.create("Results") # The results
```

```
dir.create("Temp")
```

To check if the subfolders were created within the main folder, just use the function **dir()**, this simple function will print in the console the name of the files that are currently in your working directory.

We can SET our working directory into one of the subfolders that we just created using the function **setwd()**

```
setwd("Results")
```

However, for practicality it is super-ultra-mega recommendable to work in the **MAIN FOLDER**, so go back to the previous folder or main folder by just using the function **setwd()**, instead of using a folder name, we will use simply two dots, yes two dots **".."**. This simple operation will return to the main folder.

```
setwd("..")
```

The importance of the question mark "?" or the help function

Maybe, the most important (at least for Jesús) function of R is **help** or **?**. Using help or the question mark, we can ask to R about almost anything (saddly we can't order pizza, yet)... so, let's practice!

```
help("logarithm")
```

Other important and useful functions in R, are: **head()**, **tail()**, **dim()**, **str**, **summary()**, **names()**, **class()**, **rm()**, **save.image**, **saveRDS() and readRDS()**, **load()**, **source()**, all these simple functions help us to understand the data with we are working.

Objects: creation and manipulation

In R you can create and manipulate different kind of data, from a simple numeric vector to complex spatial and/or phylogenetic data frames. The main six kinds of objects that you can create and manipulate in R, are: vector, factor, matrix, data frame, list and functions.

So, let's start with the first object, the **Vector**.

Vector

Vectors are the basic object in R and basically, contains elements of the same type (e.g., numbers, characters). Within vector exist three types: numeric, character and logic.

Numeric vector **IMPORTANT** R is case sensitive, so you need to pay attention when you name the objects.

```
a <- 10 # numeric value

b <- c(1, 2, 3, 4, 5) # numeric vector
```

```r
class(b) # ask to R which type of object is b

seq_test <- seq(from = 1, to = 20, by = 2) # Here is a sequence of numbers
# from 1 to 20, every two numbers

x = seq(10, 30) # This is a sequence from 10 to 30.
# What is the difference with the previous numeric vector?

sample(seq_test, 2, replace = T) # Sort two numbers within the object seq_test

rep_test <- rep(1:2, c(10, 3)) # Repeat the number one, ten times and the number 2
# three times

ex <- c(1:10) # Create a sequence of 1 to 10

length(ex) # Length of the object example

aa <- length(ex) # What we are doing in here?

str(seq_test) # Look at the structure of the data
```

Character vector We can also create vector of characters, which mean that instead of storing numbers we can store characters.

```r
research_groups <- c(Jeannine = "Plants", Jesus = "Birds", Maria = "Plants")

research_groups
```

```
## Jeannine    Jesus    Maria
## "Plants"  "Birds" "Plants"
```

```r
str(research_groups) # explore the character vector
```

```
##  Named chr [1:3] "Plants" "Birds" "Plants"
##  - attr(*, "names")= chr [1:3] "Jeannine" "Jesus" "Maria"
```

You can try to create a different character vector, for example, using the names of your peers.

Logic vector This kind of vector is super useful when the purpose is to create or build functions. The elements of a logic vector are **TRUE, FALSE, NA** (not available).

```r
is.factor(ex) # It is a factor? (FALSE)
```

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

```r
is.matrix(ex) # It is a matrix? (FALSE)
```

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

```r
is.vector(ex) # It is a vector? (TRUE)
```

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

```r
a < 1   # 'a' is lower than 1? (FALSE)
```

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

```r
a == 1   # 'a' is equal to 1? (TRUE)
```

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

```
a >= 1   # 'a' is higher or equal to 1? (TRUE)
```

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

```
a != 2   # the object 'a' is different of two? (TRUE) (!= negation)
```

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

Factor

A factor is useful to create categorical variables, that is very common in statistical analyses, such as the Anova.

```
data <- factor(c("small", "medium", "large"))
```

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

Matrix

A matrix is bidimensional arrangement of **vectors**, where the vectors need to be of the same type, that is, two or more numeric vectors, or two or more character vectors.

```
matx <- matrix(1:45, nrow = 15)
rownames(matx) <-  LETTERS[1:15] # names of the rows
colnames(matx) <- c("Sample01", "Sample02", "Sample03") # names of the columns or headers
```

```
matx # Inspect the matrix
```

```
##   Sample01 Sample02 Sample03
## A        1       16       31
## B        2       17       32
## C        3       18       33
## D        4       19       34
## E        5       20       35
## F        6       21       36
## G        7       22       37
## H        8       23       38
## I        9       24       39
## J       10       25       40
## K       11       26       41
## L       12       27       42
## M       13       28       43
## N       14       29       44
## O       15       30       45
```

```
class(matx) # Ask, which kind of data is?
```

```
## [1] "matrix" "array"
```

```
matx[, 1] # We can use brackets to select a specific column
```

```
##  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
matx[1, ] # We can use brackets to select a specific row
```

```
## Sample01 Sample02 Sample03
##        1       16       31
```

```
head(matx)
```

```
##   Sample01 Sample02 Sample03
## A        1       16       31
## B        2       17       32
## C        3       18       33
## D        4       19       34
## E        5       20       35
## F        6       21       36
```

```
tail(matx)
```

```
##   Sample01 Sample02 Sample03
## J       10       25       40
## K       11       26       41
## L       12       27       42
## M       13       28       43
## N       14       29       44
## O       15       30       45
```

```
#fix(matx)
str(matx)
```

```
##  int [1:15, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:15] "A" "B" "C" "D" ...
##   ..$ : chr [1:3] "Sample01" "Sample02" "Sample03"
```

```
summary(matx) # summary statistics of the data in the matrix
```

```
##     Sample01       Sample02       Sample03
##  Min.   : 1.0   Min.   :16.0   Min.   :31.0
##  1st Qu.: 4.5   1st Qu.:19.5   1st Qu.:34.5
##  Median : 8.0   Median :23.0   Median :38.0
##  Mean   : 8.0   Mean   :23.0   Mean   :38.0
##  3rd Qu.:11.5   3rd Qu.:26.5   3rd Qu.:41.5
##  Max.   :15.0   Max.   :30.0   Max.   :45.0
```

In general, when we are exploring our data for example using **head()** the function will return only the 6 first rows of our matrix, however, we can add another argument into the function. For example, **head(matx, 10)**, just add the number 10 after the comma and is possible to see the first 10 lines. This simple operation is useful specially when our matrix is large **>100 rows**.

Data frame

The difference between a matrix and a data frame is that a data frame can handle different types of vectors. You can explore more about the data frames asking R **?data.frame**. Let's create a data frame and explore its properties.

```
df <- data.frame(species = c("rufus", "cristatus", "albogularis", "paraguayae"), habitat = factor(c("fo
```

```
class(df)
```

```
## [1] "data.frame"
```

```
matx2 <- as.data.frame(matx) # We can also transform our matrix to a data frame
class(matx2)
```

```
## [1] "data.frame"
```

```
str(df)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ species : chr  "rufus" "cristatus" "albogularis" "paraguayae"
##  $ habitat : Factor w/ 4 levels "forest","savanna",..: 1 2 4 3
##  $ high    : num  10 2 7 4
##  $ distance: num  3 9 5 6
```

```
#fix(df)
#edit(df)
```

List

The list is an object that consists of an assembly of objects sorted in a hierarchical way. Here we will use the data previously created.

```
lst <- list(data, df, matx)
```

```
str(lst)
```

```
## List of 3
##  $ : Factor w/ 3 levels "large","medium",..: 3 2 1
##  $ :'data.frame':    4 obs. of  4 variables:
##   ..$ species : chr [1:4] "rufus" "cristatus" "albogularis" "paraguayae"
##   ..$ habitat : Factor w/ 4 levels "forest","savanna",..: 1 2 4 3
##   ..$ high    : num [1:4] 10 2 7 4
##   ..$ distance: num [1:4] 3 9 5 6
##  $ : int [1:15, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:15] "A" "B" "C" "D" ...
##   .. ..$ : chr [1:3] "Sample01" "Sample02" "Sample03"
```

```
class(lst)
```

```
## [1] "list"
```

Now, inspect the objects that are stored into our object **lst**. To do this, we just need to use two brackets [[]].

```
lst[[1]]
```

```
## [1] small  medium large
## Levels: large medium small
```

```
lst[[2]]
```

```
##       species    habitat high distance
## 1       rufus     forest   10        3
## 2   cristatus    savanna    2        9
## 3 albogularis      urban    7        5
## 4  paraguayae transition    4        6
```

```
lst[[3]]
```

```
##   Sample01 Sample02 Sample03
## A        1       16       31
## B        2       17       32
## C        3       18       33
## D        4       19       34
```

```
## E        5       20       35
## F        6       21       36
## G        7       22       37
## H        8       23       38
## I        9       24       39
## J       10       25       40
## K       11       26       41
## L       12       27       42
## M       13       28       43
## N       14       29       44
## O       15       30       45
```

At this point we explored the most common objects in R, understanding the structure of each class of object (from vectors to lists) is maybe the most important step to learn R.

Install and load packages

Although R is a programming language, it is also possible to use different auxiliary packages that are available for free to download and to install in our computers. Install new packages into R is easy and just needs a simple function **install.packages()**. For more information of how to install new packages, you just need to ask R, using **?install.packages**

```
install.packages("PACKAGE NAME")
```

The reverse function is **remove.packages()**.

Most of the time, we do not remember if we already have a package installed in our computer, so, if we are tired and do not want to go to our R folder packages and check if the package is in fact installed, we can use the next command.

```
if ( ! ("PACKAGE NAME" %in% installed.packages())) {install.packages("PACKAGE NAME", dependencies = T)}
```

To load an installed package you can just type, **library() or require()**

```
library("PACKAGE NAME")
require("PACKAGE NAME")
```

Sometimes we need to install a lot of packages and installing one by one will require time and patience, which most of the time we don't have Lol. To solve that issue we can create a vector with the names of the packages and create a simple function that help us to install R with just one click!

Package names

packages <- c("ggplot2", "scales", "cowplot", "dplyr", "tidyr", "viridis")

Install packages not yet installed

installed_packages <- packages %in% rownames(installed.packages()) if (any(installed_packages == FALSE)) { install.packages(packages[!installed_packages]) }

Packages loading

invisible(lapply(packages, library, character.only = TRUE))

R as a calculator

R can be used as a calculator, for example, we can use the information created before to make some arithmetic operations.

```
b[4]+seq_test[10]
```

```
## [1] 23
```

```
b[4]*seq_test[10]
```

```
## [1] 76
```

```
seq_test[5]/df[3, 3]
```

```
## [1] 1.285714
```

```
matx[, 3][4]-df[4, 4]
```

```
##  D
## 28
```

```
seq_test^7
```

```
##  [1]         1      2187     78125    823543   4782969  19487171  62748517
##  [8] 170859375 410338673 893871739
```

```
seq_test*7
```

```
##  [1]   7  21  35  49  63  77  91 105 119 133
```

```
seq_test+7
```

```
##  [1]  8 10 12 14 16 18 20 22 24 26
```

```
seq_test-7
```

```
##  [1] -6 -4 -2  0  2  4  6  8 10 12
```

```
mean(seq_test)
```

```
## [1] 10
```

```
max(seq_test)
```

```
## [1] 19
```

```
min(seq_test)
```

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

```
sum(seq_test)
```

```
## [1] 100
```

```
log(seq_test)
```

```
##  [1] 0.000000 1.098612 1.609438 1.945910 2.197225 2.397895 2.564949 2.708050
##  [9] 2.833213 2.944439
```

```
sqrt(seq_test)
```

```
##  [1] 1.000000 1.732051 2.236068 2.645751 3.000000 3.316625 3.605551 3.872983
##  [9] 4.123106 4.358899
```

```
cor(matx[, 1], matx[, 2])
```

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

Data import/export

As indicated before, in R you can handle different information (from vector to data frames) and basically most of our data is stored in an Excel spreadsheet or in files that have the extension of **.csv** (comma-separated values file) or **.txt** (Text X Text or text file that contains unformatted text).

Most of these files are imported in R are **data frames**, but, as we were practicing, we now have the tools to handle or transform the information into different objects.

The function to import data to R is simple **read.table()** or **read.csv()**, and using these simple functions, you can import the data and transform it in other kind of objects So, lets practice!

```
dat <- read.table("Data/Sample.txt")

dat2 <- read.table("Data/Sample.txt", row.names = 1, header = TRUE)

dat3 <- read.csv("Data/Sample.csv")
```

```
class(dat)
```

```
## [1] "data.frame"
```

```
class(dat2)
```

```
## [1] "data.frame"
```

```
class(dat3)
```

```
## [1] "data.frame"
```

```
dat3Sample <- dat3[1:50, 1:4]
dim(dat3Sample)
```

```
## [1] 50  4
```

```
dat4 <- na.omit(as.matrix(read.csv("Data/Sample.csv", row.names = 1, header = TRUE)))
class(dat4)
```

```
## [1] "matrix" "array"
```

```
head(dat4, 10)
```

```
##     Longitude.x. Latitude.y.        PD SR      PSVs SR.1        vars     aMRD
## 192       -108.5        26.5 0.3789908  3 0.7965072    3 0.004459021 17.66667
## 222        -78.5        25.5 0.2167810  2 0.3536695    2 0.012361281 12.50000
## 229       -107.5        24.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 230       -106.5        24.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 235       -101.5        24.5 0.2789230  2 0.8175072    2 0.012361281 19.50000
## 245       -106.5        23.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 246       -105.5        23.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 247       -104.5        23.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 248       -103.5        23.5 0.2684503  2 0.7393375    2 0.012361281 18.50000
## 250       -101.5        23.5 0.2789230  2 0.8175072    2 0.012361281 19.50000
##         aMDR      aAGES
## 192 14.27362 0.05628246
```

```
## 222 26.88631 0.01707297
## 229 14.49822 0.05489227
## 230 14.49822 0.05489227
## 235 17.25644 0.05057945
## 245 14.49822 0.05489227
## 246 14.49822 0.05489227
## 247 14.49822 0.05489227
## 248 14.49822 0.05489227
## 250 17.25644 0.05057945
```

```
dat4[1:20, 1:4]
```

```
##     Longitude.x. Latitude.y.        PD SR
## 192       -108.5        26.5 0.3789908  3
## 222        -78.5        25.5 0.2167810  2
## 229       -107.5        24.5 0.2684503  2
## 230       -106.5        24.5 0.2684503  2
## 235       -101.5        24.5 0.2789230  2
## 245       -106.5        23.5 0.2684503  2
## 246       -105.5        23.5 0.2684503  2
## 247       -104.5        23.5 0.2684503  2
## 248       -103.5        23.5 0.2684503  2
## 250       -101.5        23.5 0.2789230  2
## 251       -100.5        23.5 0.3817156  3
## 252        -99.5        23.5 0.3817156  3
## 255        -82.5        23.5 1.2950881 14
## 256        -81.5        23.5 0.5936688  5
## 257        -80.5        23.5 0.2961977  2
## 261       -106.5        22.5 0.2684503  2
## 262       -105.5        22.5 0.2684503  2
## 263       -104.5        22.5 0.2684503  2
## 264       -103.5        22.5 0.2684503  2
## 265       -102.5        22.5 0.2684503  2
```

You can also import your data using the same functions, but without specifying the address. Notice that we do not recomend this procedure as you can't control the **directory structure**, but is useful when you just are exploring data.

You can also save your data from R using the function **write.table** or **write.csv**. Lets save the dat3Sample. Notice that always we need to specify the correct address, in our case we will save the data in the subolder **Data**.

```
is.na(dat3Sample)
write.csv(dat3Sample, file = "Data/dat3Sample.csv")
```

Phylogenetic data

To study biodiversity is important to first understand the data and one common data used now is the phylogentic data or phylogenetic trees that describe the evolutionary relationships between and among lineages. From here until the end of this short tutorial we will try to explain the basics of how to import/export and handle phylogenetic information. You can find extra information at https://www.r-phylo.org/wiki/HowTo/Table_of_Content e http://www.mpcm-evolution.org/practice/online-practical-material-chapter-2.

Formats

The two most common formats in which the phylogenies are stored are the Newick and Nexus (Maddison et al. 1997).

The Newick format represent the phylogenetic relationships as **"(", "," and ":"**, so the species relationships can be represented as follows:

```
((A:10,B:9)D:5,C:15)F;
```

Using this notation, the parenthesis links the lineages to a specific node of the tree and the comma **","** separates the lineages that descend from that node. The colon punctuation **":"** can be used after the name of the node and the subsequent numeric values represent the branch lenght. Finally, the semicolon punctuation **";"** indicate the end of the phylogenetic tree.

Now we can see how this format works, but first, check if we have the R packages for this purpose.

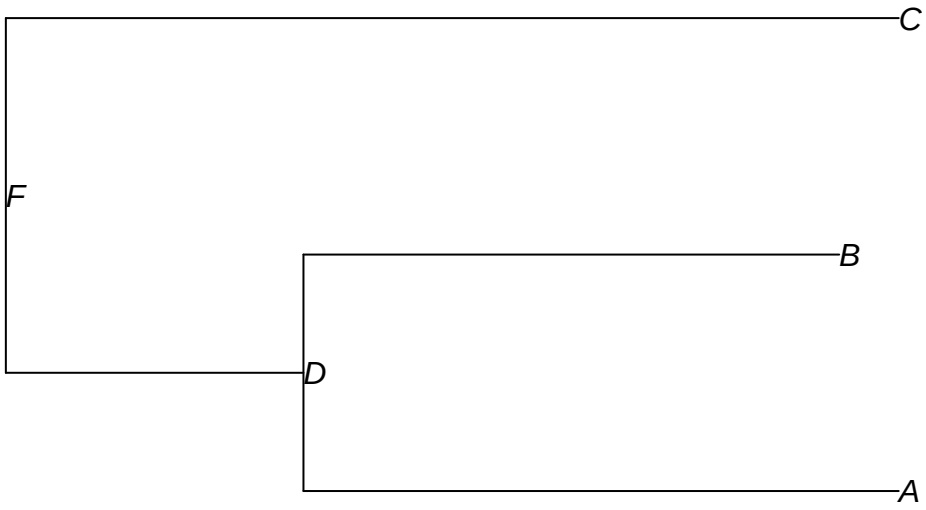
```
if ( ! ("ape" %in% installed.packages())) {install.packages("ape", dependencies = T)}
```

```
require(ape)
```

```
## Here we will create a phylogenetic tree in Newick format
newick_tree <- "((A:10,B:9)D:5,C:15)F;"

## Read the tre
newick_tree <- read.tree(text = newick_tree)

## And now we can plot the phylogentic tree
plot(newick_tree, show.node.label = TRUE)
```



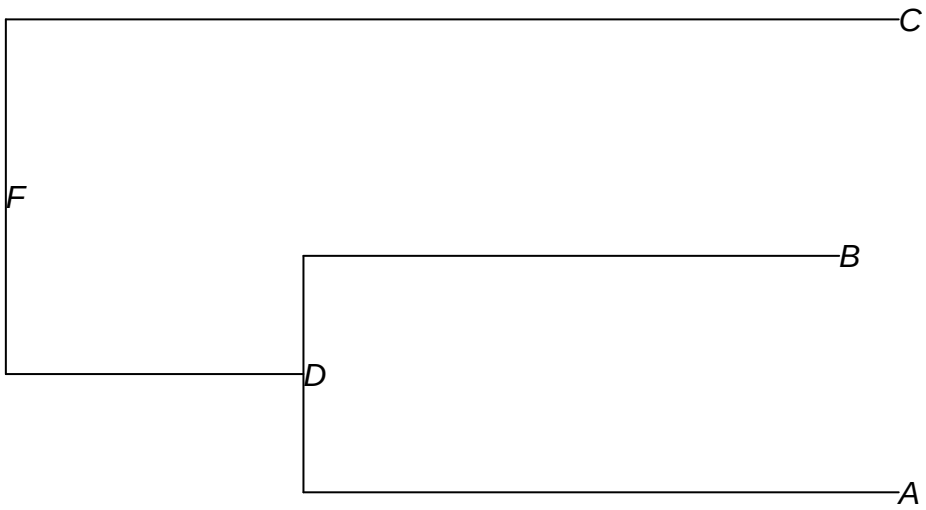
The other format is the **Nexus**, and after some time using it, we can say the Nexus format have more flexibility for working. An example of a Nexus format is as follow:

```
#NEXUS
BEGIN TAXA;
DIMENSIONS NTAXA=3;
TaxLabels A B C;
END;
BEGIN TREES;
TREE=((A:10,B:9)D:5,C:15)F;
END;
```

```
## First create a Nexus file in the working directory
cat(
 "#NEXUS
 BEGIN TAXA;
 DIMENSIONS NTAXA=3;
 TaxLabels A B C;
 END;
 BEGIN TREES;
 TREE=((A:10,B:9)D:5,C:15)F;
 END;",
file = "Data/Nexus_tree.nex"
)
```

```
## Now read the phylogenetic tree, but look that instead of using read.tree we are using read.nexus
nexus_tree <- read.nexus("Data/Nexus_tree.nex")
```

```
## lets plot the example
plot(nexus_tree, show.node.label = TRUE)
```



Now lets inspect our phylogenetic trees.

```
str(nexus_tree)
```

```
## List of 5
##  $ edge       : int [1:4, 1:2] 4 5 5 4 5 1 2 3
##  $ edge.length: num [1:4] 5 10 9 15
##  $ Nnode      : int 2
##  $ node.label : chr [1:2] "F" "D"
##  $ tip.label  : chr [1:3] "A" "B" "C"
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

```
nexus_tree$tip.label
```

```
## [1] "A" "B" "C"
```

If we want to know about the branch lenght of the tree we just need to select **edge.lenght**

```
nexus_tree$edge.length
```

```
## [1]  5 10  9 15
```

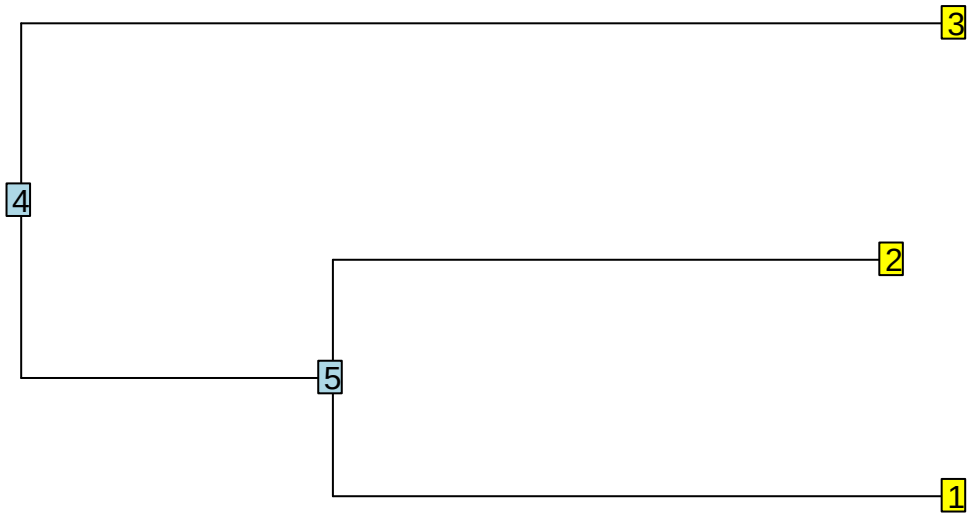
An important component of a phylo object is the matrix called **edge**. In this matrix, each **row** represents a **branch** in the tree and the **first column** shows the index of the ancestral node of the branch and the **second column** shows the descendant node of that branch. Lets inspect!

```
nexus_tree$edge
```

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

We know, it is a little hard to follow even with small trees as the example, but, if we plot the phylogenetic tree, the information within it is easier to understand.

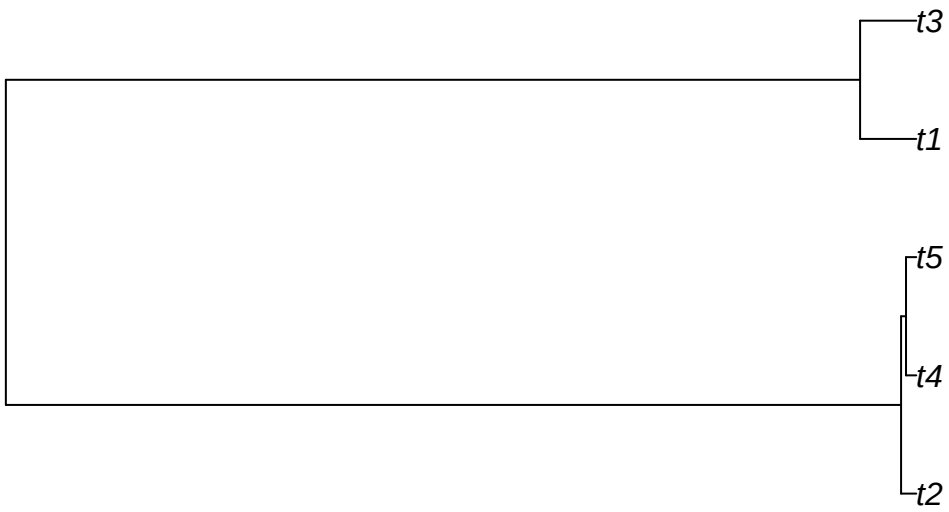
```
# Lets plot the tree
plot(nexus_tree, show.tip.label = FALSE)
# Add the internal nodes
nodelabels()
# Add the tips or lineages
tiplabels()
```



Finally, the phylogenies can also be imported in form of a list and in phylogenetic comparative methods this list of phylogenies is called **multiPhylo**, and we can import/export these multiPhylos in the two formats.

```
# Simulate 10 phylogenies, each one with 5 species
multitree <- replicate(10, rcoal(5), simplify = FALSE)
# Store the list of trees as a multiPhylo object
class(multitree) <- "multiPhylo"

# Plot a single tree from the 10
plot(multitree[[10]])
```




```
par(mfrow = c(2, 2))
plot(multitree[[1]])
plot(multitree[[3]])
plot(multitree[[7]])
plot(multitree[[10]])
```


```
# Exporting the phylogenies as a single Newick file.
write.tree(phy = multitree, file = "Data/multitree_example_newick.txt")
multitree_example_newick <- read.tree("Data/multitree_example_newick.txt")
multitree_example_newick
```

```
## 10 phylogenetic trees
```

```
# Exporting the phylogenies as a single Nexus file.
write.nexus(phy = multitree, file = "Data/multitree_example_nexus.nex")
multitree_example_nexus <- read.nexus("Data/multitree_example_nexus.nex")
multitree_example_nexus
```

```
## 10 phylogenetic trees
```

Gentle intro to loops

In programming one of the most important tool is the **loop** AKA **for**. Basically, a loop runs for **n** number of steps in a previously defined statement.

The basic syntax struture of a loop is:

```
for (variable in vector) {
  execute defined statements
}
```

When we are programming it is common to use the loop variable **i** to determine the number of steps. Why not other letter?, well **i** is the first letter of the word **iteration** —duh! Anyway, you can use any letter or word as a loop variable.

So, lets take a look.

```
for (i in 1:10){
  cat(i, sep = '')
}
```

```
## 12345678910
```

Notice that the number of steps is determined by the loop variable and in this example is a sequence of steps from 1 to 10, that correspond to the second element of the **for loop**, the **vector**.

You can modify the previous statement to obtain different results, for example:

```
for (i in 1:10){
  cat(i, sep = '\n')
}
```

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

Or using a previous object:

```
for (i in 5:length(ex)){
  cat(i, sep = '\n')
}
```

```
## 5
## 6
## 7
## 8
## 9
## 10
```

Or to make calculations

```
for (i in 5:length(ex)){
  b2 <- b^2
  b3 <- b*2
```

```
  b4 <- b+10
}
```

```
BioSciNames <- c("Jeannine", "Jesús", "Grant", "Lauren", "Ashley", "Efrata", "Calista",
                 "Madeline", "Andrew", "Addie", "Robin", "Aidan", "Jocelyn", "Marin",
                 "Erin", "Abigail", "Luke", "Rose", "Kyle", "Adoree", "Jessica",
                 "Gayatri", "Lucy", "Francesca")
```

```
for (i in 3:length(BioSciNames)){
  cat("Hi,", BioSciNames[i], ", welcome to the first practice!", "\n");
}
```

```
## Hi, Grant , welcome to the first practice!
## Hi, Lauren , welcome to the first practice!
## Hi, Ashley , welcome to the first practice!
## Hi, Efrata , welcome to the first practice!
## Hi, Calista , welcome to the first practice!
## Hi, Madeline , welcome to the first practice!
## Hi, Andrew , welcome to the first practice!
## Hi, Addie , welcome to the first practice!
## Hi, Robin , welcome to the first practice!
## Hi, Aidan , welcome to the first practice!
## Hi, Jocelyn , welcome to the first practice!
## Hi, Marin , welcome to the first practice!
## Hi, Erin , welcome to the first practice!
## Hi, Abigail , welcome to the first practice!
## Hi, Luke , welcome to the first practice!
## Hi, Rose , welcome to the first practice!
## Hi, Kyle , welcome to the first practice!
## Hi, Adoree , welcome to the first practice!
## Hi, Jessica , welcome to the first practice!
## Hi, Gayatri , welcome to the first practice!
## Hi, Lucy , welcome to the first practice!
## Hi, Francesca , welcome to the first practice!
```

We have covered basic aspects of R, from exploring and managing object to import/export data and basics into loops. We hope that this short tutorial can be useful not only for the **Biodiversity Science** course, but for your specific projects. Remember, practice, practice, practice!

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997). NEXUS: An Extensible File Format for Systematic Information. Systematic Biology 46, 590.