Hands-on training 1: Water resources

Goals

- Introduction to R coding
- Basic R operations especially using raster and ggplot2 packages
- Simple raster calculations
- Basics of illustrations

Data

- Monthly precipitation and temperature: WorldClim, interpolations of observed data, representative of 1950-2000. www.worldclim.org (data under climate change for year 2050 available in supporting materials)
- Runoff: WATCH mean annual runoff and monthly runoff, 1971-2000. www.eu-watch.org
- Countries: Natural Earth, free vector and raster map data at https://naturalearthdata.org
- Basins: Hydro1K, https://doi.org/10.5066/F77P8WN0

Quick reference

Remember to create a course folder under your network home directory. NOT on the local disk!

General syntax

R basic syntax is very simple. In today's hands-on training, it is almost enough to know that you can put commands (internal commands, assignments, function calls) each on their own line.

```
a <- 10           # this assigns a value to a variable
b <- a * 2        # this uses the previously set variable
# and as you noticed, comments start with a hashtag
```

In R, the assignment operator (<-) is reserved for assigning values for variables. It is possible, however strongly discouraged to use = in assignment because = is reserved for giving parameter values for functions (more of this later). So, always use only <- when assigning values to variables.

Setting your working directory

```
setwd("path to your directory") # set working directory to a folder
getwd()                         # get current working directory
list.files()                    # list files in current working directory
```

If no absolute paths are given in for example file read calls, R searches for the files or relative paths in the current working directory. Therefore, it is important to know where the data resides and keep the working directory in a fixed folder.

File management

```
load("variables.Rdata")                          # load R variables from a file
save(variables, file = "new_variables.Rdata")   # save variables to a file
```

`load` and `save` read from and write to from the current working directory if no other absolute path is given. These functions work with R variables which cannot be read by for example Excel. Various alternatives exist for outputting tabular and other kinds of data, of which we use at least `write.csv` and `ggsave` in this week's exercise.

```
write.csv(table, file = "table.csv")   # write a variable into a csv file
```

Operators <-, = and ==

As noted, <- is reserved for assigning variables while = is used in function calls to define parameters (like before in `write.csv` and `save`) Further, == denotes comparison, which returns TRUE or FALSE depending on the test result.

```
variable <- 0.5  # assignment to a variable
variable == 0 # comparison, returns FALSE but does not assign anything
output <- some_function(x = variable)  # a function call to some_function()
```

Logical variables can also be created:

```
example_logical = a == b
```

example_logical will equal TRUE if a equals b, and FALSE if a does not equal b. If either of the variables is NA, the result will also be NA. Remember that variables can also be vectors and arrays, so can logical variables!

Accessing list and array elements

In R, there are at least vectors (1D), matrices (2D), and arrays (nD) for solely numeric/character data, data frames for mixed type data (2D) and lists for storing multiple objects in one variable. Indexing starts from one.

```
vector <- c(1,2)                                  # initialize a vector
matrix <- matrix(c(vector, vector), nrows = 2)  # initialize a matrix
array <- array(data = matrix, dim = c(2,2,2))   # initialize an array
df <- data.frame(vector)                          # initialize a data frame
list <- list(vector, matrix, array, df)          # initialize a list
names(list) <- c("vec", "mat", "arr", "df")     # set names of list elements

vector[1]          # first element of a vector
matrix[2,3]        # cell in the second row and the third column of a matrix
matrix[,1]         # the whole first column of a matrix (returns a vector)
matrix[5,]         # the fifth row of a matrix
array[4,5,7]       # row 4, column 5, index 7 in the 3rd dimension of an array
array[,,3]         # index 3 in the 3rd dimension, returns a matrix

df$vector              # a column named "vector" in a data frame
list$vec               # a member named "vec" in a list
df[1,]                 # the first row of a data frame (same as matrix)
list[[3]]              # the third member of a list (single member)
list[c(2,4)]           # the second and fourth members of a list
```

Functions to know

Help for any function is available in RStudio by giving a question mark and the name of the function in the command window, for example: `?raster`. Typically, the help window contains also some examples on how to use that particular function.

```
sum(vector)                    # sum the elements of a vector
sum(vector, na.rm = TRUE)      # sum the elements, remove NA values
unique(vector)                 # find unique values in a vector
apply(...)                     # apply something over a matrix/an array
lapply(...)                    # apply something over a list
raster(matrix, ...)            # initialize a raster
```

Loops and conditional statements

You will need for loops, which allow you to repeat a group of commands a known number of times

```
for (index_variable in startvalue:endvalue){

    commands     # here you can use index_variable for
                 # calculations but don't change it!

}
```

and even if not necessarily today, soon enough you will need the conditional statement:

```
if (test){       # test can be anything that gives logical true or false
    commands     # only perform the commands if the test was true
} else {
    commands     # only perform the commands if the test was false
}
```

Graphics

Graphics can be done in multiple ways with R. Quite many widely used packages, such as the raster package, have an internal function for plotting objects. However, those functions are often very limited, which is why we are using ggplot2 package to create nice looking graphs and to export them for further editing with Adobe Illustrator.

```
plot(raster, ...)                  # plot a raster
ggplot(data) +                     # create a scatter plot
    geom_point(aes(x = x_values,
                   y = y_values))
```