5. Functions

Functions are useful in programming because they allow you to encapsulate reusable code, making your code more readable, and maintainable. They help reduce repetition by letting you define a task and reuse it with varying input parameters, improving efficiency and reducing the chance of errors.

5.1 Defining functions

The general structure to define a function in R is

```
Name_of_function <- function(names of input argument(s))
{
    ... commands to be executed ...
    return(output)
}
```

For each input argument one can specify default values. For instance, if a function "Test" has two input parameters `a` and `b` one could, set the default 0 for `b` as follows

```
Test <- function(a,b=0)
{
    ... commands to be executed ...
    return(output)
}
```

Analogously, one could set a default for `a` or for both, `a` and `b`.

Once the function definition has been "executed" (compiled/loaded), one can use it with specific input arguments.

For the function `Test()` above with two default values, on may call it, for instance, with

```
  Test(10,20)      # executes Test(a,b) with a=10, b=20
  Test(b=20,a=10)  # executes Test(a,b) with a=10, b=20
  Test(a=10)       # executes Test(a,b) with a=10, b=0 (works because we set b=0 as default)
```

Example:

In "Rcourse_4.R" and "RCourse_4.pdf" we wrote some code that calculated which converts the temperature of a unit from Celsius, Kelvin and Fahrenheit into the other units. To use this code more efficiently we could wrap it into a function.

```
Temperature <- function(Temp=30, Scale="Fahrenheit"){
  if (substr(Scale,1,1)=="F" | substr(Scale,1,1) == "f"){
    TempF <- Temp
    TempC <- (Temp - 32)*5/9
    TempK <- TempC + 273.15
  } else if (substr(Scale,1,1)=="C" | substr(Scale,1,1) == "c"){
    TempC <- Temp
    TempF <- 32 + 9*Temp/5
    TempK <- Temp + 273.15
  } else {
    TempK <- Temp
    TempC <- Temp - 273.15
    TempF <- 32 + 9*TempC/5
  }
  return(c('Celsius'=TempC,'Fahrenheit'=TempF,'Kelvin'=TempK))
}
```

Then we can use the function `Temperature` as follows

```r
Temperature(Scale='fahrensomething',Temp=80)
```

```
##    Celsius Fahrenheit     Kelvin
##   26.66667   80.00000  299.81667
```

```r
Temperature(Temp=273.15,Scale='Kelvin')
```

```
##    Celsius Fahrenheit     Kelvin
##       0.00      32.00     273.15
```

Exercise:

Write the code from RCourse_4.R to calculate the Fibonacci series in a function. Try to catch the invalid input parameters.

5.2 Organizing Code when working with functions

Having both functions and executable commands in the same script file is often chaotic. It Therefore, makes sense to store the functions in a separate script file. Thus, a project is often separated in two (or more) script files as follows

- script file: containing a collection of executable commands plus comments (!)
- function file: containing the code of one or several functions plus comments (!)

There are two ways to compile/load and use a function: - Execute its source code. - Run the command `source('filename.R')` to execute all commands in filename.R', in particular, all functions within that file.

Then one can use the function(s) within the R console or script files during the current session.

Example:

Have a look at the script file "DNA_Functions.R". We can now source and use those functions in a separate script file.

```r
source("DNA_Functions.R") # source function file

sequence <- "ATGCGATCGATCGATCGTAGCTAGCTAGCTAGC"
GC_content(sequence)
```

```
## [1] 51.51515
```

```r
DNA_to_RNA(sequence)
```

```
## [1] "AUGCGAUCGAUCGAUCGUAGCUAGCUAGCUAGC"
```

5.3 Predefined functions in R

R comes with a wide range of predefined functions that are essential for data analysis and manipulation. Some of those you have already encountered in the previous chapters. Some of the most frequently used R functions include:

- `plot()` for data visualization
- `c()` for combining values into a vector
- `length()` to get the number of elements in a vector
- `sum()` to add all the elements of a vector
- `exp()` to calculate the exponential of a number or a vector (component-wise)
- `read.table()` to read a dataset in “txt”-form into R.