BASICS OF PROGRAMMING USING R

0) Bit (short for binary digit): is the smallest unit of data in a computer.
A bit has a single binary value, either 1 or 0.

Bits are used to store data and execute instructions in bit multiples (bytes).

Byte: a group of binary digits or bits (ones and zeros) (usually eight)

Bytes defined a word or a value for a type of variable (see below). E.g.,
Integer values are usually stored in four (4) bytes.

Task: what will be the maximum value for an integer number?

1) What is a text editor (script), a compiler? and how are instructions read
by a computer?

Script: a list of commands that are executed by a certain program

Compiler: a special program that processes statements written in a particular programming language (e.g., c/c++, Java, python, r ...) and turns them into machine language that a computer processor uses

2) Type of variables

Integer number (Int): ..., -1, 0, 1, ...

Floating.point number (float): 1,33 ...

Character (char): "a"

String: "hello world"

Boolean: 1 or 0; true or false

3) Operators: are symbols used to perform operations on operands

e.g., 2 + 3; here + is an operator to carry out an addition operation on the operands 2 and 3

3.0) Use the symbol # to write comments. The information after this symbol will be ignored by the compiler.
Example:

this is a comment

It is good practice to comment (i.e., clearly briefly explain) the code

such that anyone (including us - the programmer - in a future) can understand it, repeat it or modify it.

3.1) Types of operators:

Assignment operators

```
x <- value
x <<- value
value -> x
value ->> x
x = value (this is the "universal" assignment operator)
```

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

3.2) function To print output on console:

print()

BONUS) Basic string functions in R

nchar() ; get the number of characters in the string

example:

```
> nchar("name")
> 4
```

to replace a pattern by anotherone into the string
sub("pattern", "replacement", stringobject)

example:

```
> sub("na", "me", "name")
> "meme"
```

gsub(); same as above, but replaces all patterns, if matches more than once

to find a specific character or string pattern in a string:
regexpr("pattern", x) # returns position of 1st match in string
gregexpr('pattern', x) # returns positions of every match

regexpr() and gregexpr() return a list. So maybe it can be useful to unlist them and transform into vectors
unlist()

example:

```
> unlist (gregexpr(':', "name: XXX; age: YY"))
> 5     15
```

Use substr() function to extract part of the string based on position in the string where first and last are the locations in the text string, usually found by the regexpr()

substr(x, first, last)

using the above example:

```
> substr("name: XXX; age: YY", 5, 15)
> ": XXX; age:"
```

4) Data Types

4.1) Vectors

```
v <- c(1, 2.5, 10)

v [2] # 2nd element in the vector  v

# in r, the first element has the index 1
```

4.2) Matrices

All columns in a matrix must have the same variable type and the same length

```
M <- matrix(1:20, nrow = 4, ncol = 5)

M[1,2]  # element in 1st row, 2nd column

M[1, ]  # all elements in 1st row
```

Tips) For basic functions on matrix algebra visit https://www.statmethods.net/advstats/matrix.html

4.3) Data frames

are more general than a matrix in that different columns
can have different variable types (e.g., number, char, etc)

```
n <- c(1, 0, 0, 1)
b <- c(TRUE, FALSE, FALSE, TRUE)
y <- c("blue", "green", "green", "blue")

myData <- data.frame(y, b, n)
names(myData) <- c("color", "boolean type 1", "boolean type 2")
```

4.4) Lists

are R objects which contain elements of different types
(e.g., numbers, strings, vectors, matrices, ...)
A list is created with list()

example:

```
# Create a list containing a vector, a matrix and a list.
myList <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
list("green",12.3))

# Give names to the elements in the list.
names(myList) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Show the list.
print(myList)

# element 2 in the list
myList[[2]]
```

5) Logical and conditional statements

5.1) If statements:

```
if ( test_expression) { statement }
```

example: `if ( x > y ) { x = x - 1 }`

5.2) If-else statements:

```
if (test_expression)
{ statement }
else {statement } ## if expression is not met, then...
```

example:

```
x <- -5

if(x >= 0)
{
        print("Non-negative number")
}
else
```

```
{
        print("Negative number")
}
```

Task: write a program that receives a number and then print
on the console whether the input number is even or uneven

5.3) multiple if-else statements:

```
if ( test_expression ) { statement }
else if ( test_expression_2 ) { statement_2 }
else { statement }
```

5.4) Switch statements

```
switch ( condition ) { statement }
```

6) Loops: cycling or iterating instructions

6.1) For-loop

test whether variable's current value is within a specified range,
and if so, execute statement

```
for (i in 1:10) # from 1 to 10, with increments of 1
{ statement }
```

For loops can be nested!

Task: create a vector of zeros of length 3
Substract 1 from its 1st and 3rd element and sum 1 to its 2nd
using a loop
print the new vector

How can you write the loop to work for any vectors length?

6.2) While-loop

```
initialization of loop-control-varible
while ( test_expression)
{
        statement
        update-value-of loop-control-variable
}
```

6.3) Repeat-loop

```
repeat
{
        statement

        if ( test-expression ) # check condition and break loop if met
```

```
        {
                statement_1
                break # instruction to break the loop
        }
        else { statement_2 }
}
```

7) Functions

```
myFunctionName <- function(arg1, arg2, ... )
{
        statements # include operations here

        return (object) # here the object is the result you want
                        # the function to return
}
```

Example:

```
myfunction <- function(x, y, SUM)
{
        if ( SUM == TRUE)
        {
                result <- x + y
        } else {
                        result <- x - y
                }

        return (result)
}

myfunction(1, 3, TRUE)
```

answer: 4

Task: Create a function of your preference

8) Working with data

8.1) Reading csv files (check documentation: ?read.csv())

```
df <- read.csv("data_example.csv", sep = ",", header = TRUE)
```

8.2) Head, structure and description of the data frame

```
# overview of first rows of the dataframe

head(df)

# data type structure of df

str(df)
```

```
        # summary description of df

        summary(df)
```

8.3) Select by column / row

```
        df[,1] # all rows in first column

        # first element (upper left corner) in data frame
        df[1,1]

        # all columns, values in first row
        df[1,]

        # select by column name
        df$column_name # check names(df)
```

The $ operator enable us to extract the information contained in a given object (in this case, the object df) under the name "column_name"

8.4) Descriptive statistics

```
        mean(df[,1]) # mean of the variable in first column

        mean(df$duration) # mean of column named "duration"

        sd(df$duration)   # computes the standard deviation

        median(df$duration) # computes the median
```

8.5) Some plotting

```
        plot(df) # columns need to be numeric! (check documentation)
                 # quick way of looking at the data and correlations
                 # among variables

        # plot only two variables (columns)
        plot(df$duration, df$maximum_frequency)

        # a nicer correlation plot between two variables

        install.packages("ggpubr") # install package ggpubr
        library(ggpubr) # load the package

        # use the package
        ggscatter(df, x = names(df)[3], y = names(df)[2]
                 ,add = "reg.line" # add a linear regression line
                 ,conf.int = TRUE  # add confidence interval
                 ,conf.int.level = 0.95 # confidence level
                 ,cor.coef = TRUE  # correlation coefficient and p-value
                 ,cor.method = "pearson" # "pearson", "kendall", or "spearman"
                )
```

```
        # Some background: correlation is a bivariate analysis that measures
        # the strength of the association between two variables, and the
direction
        # of the relationship.
        # Typically, the value of the correlation coefficient varies between
        # (-1 and 1)
        # Remember to check and test the assumptions required by the
        # statistical methods used

####################

        REQUIRED LIBRARIES FOR STEP 2

 install.packages("stats") # kmeans model
 install.packages("cluster") # silhouette coefficient function
 install.packages("ggplot2")
 install.packages("FactoMineR")
 install.packages("factoextra")
 install.packages("rgl") # 3d plots
 install.packages("BBmisc")# normalize() function
 install.packages("e1071") # sigmoid function
 install.packages("car") # scatter3d plot
```