**F.FABRI**
**Practical Tests (R Software)**
**Promotion 2019 - ISAFE**

# Programming with R: datasheet 1
# The basics of R programming computing

I Usual operations

| Operator | Descriptions |
|---|---|
| + | Addition |
| - | Substraction |
| * | Multiplication |
| / | Division |
| ^ *or* ** | Exponential |
| *x %% y* | Modulo (x mod y) |
| *x %/% y* | Whole part of the division |

**Applications :**

1) Calculate : 7 + 8.1
2) Calculate : $2^{\frac{1}{2}} + 2^{10}$
3) What is the result of the Euclidean division of 57 by 7?

II Logical operators

| Operators | Descriptions |
|---|---|
| < | Less than |
| <= | Less than or equal to |
| > | Superior |
| >= | Greater than or equal to |
| == | Exactly equal to |
| != | Different |
| *x \| y* | x or y |
| *x & y* | x and y |

**Applications :**

What do the following expressions give:

1) 7 > 8
2) 7 <= 9 <= 5 (Why is there an error message?)
3) 7 <= 9 | 9 <= 5
4) 7 <= 9 & 9 <= 5
5) pi
6) 3.141593 == pi
7) cos(pi) != 1

III Vector manipulation

| Fonctions | Descriptions |
|---|---|
| *x = c(1,2,4,8)* | Assignment of the vector (1,2,4,8) to the variable x |
| *y = c($v_1$,$v_2$,...)* | Concatenation of vectors v1, v2, ... and assignment to variable y |
| *length(x)* | Number of elements of vector x |
| *x[k]* | Returns the kth element of vector x |
| *x[v]* | Return the elements of vector x whose indices correspond to vector v |
| *x[-k]* | Remove the kth element from the vector *x* |
| *x[-v]* | Removes the elements from vector x whose indices correspond to vector v |
| *which(x == k)* | Return the indices of the elements of the vector x equal to *k* |
| *rank(x)* | Returns the ranks of the elements of the vector x sorted by order (ascending for numeric values, alphabetical for characters) |
| *sort(x)* | Returns the vector x sorted by order (ascending for numeric values, alphabetical for characters) |
| *sum(x)* | Returns the sum of the elements of x (if x is a vector of digital elements) |
| *names(x)* | Name all the elements of vector x |
| *identical($x_1$,$x_2$,...)* | Return TRUE if all the objects x1, x2, ... are exactly the same and FALSE are different |
| *all.equal($x_1$,$x_2$,...)* | Returns TRUE if there is an "approximate equality" of the objects x1, x2 ,… otherwise it displays a summary of the differences |

**Applications:**

1. Assign the elements 1, 2, 4, 7, 8, 16, 32 to the vector V.
2. What is W value if we type the command *W = c (V, V)*?
3. What is the index of element 7 in vector V?
4. Remove element 7 from vector V and assign the new vector to V.
5. What is *V [4]* value?
6. What is the result of *V + 1*? What is the result of *V * 2*? Does this surprise you?
7. What is the result of *V + c (1,2)*? and *V * c (2,3)*?
8. What does operation *V% *% V* do? Does the presentation seem changed to you?
9. What is the result for the: *names (V)* command?
10. Enter the command: names *(V) = c ("a", "b", "c", "d", "e")*. What is the result for the command: *names (V)*?
11. What is the sum of the elements of *V* value?
12. Assign to *W* the elements of *V* greater than 5.
13. To remove the names of the elements from *W*, we can perform the command: *names (W) = c ()*.
14. What do you notice when prompting for a = c ("Give me", 1, "correct answer") and print (a)?
15. Enter the command: *a [2] + 1*
16. Try again with the command: *as.numeric* (a [2]) + 1
17. Test the identical and *all.equal* functions. What differences do you notice between identical (0.9,1.1-0.2) and *all.equal* (0.9,1.1-0.2)?

IV Data generation

| Functions | Descriptions |
|---|---|
| *seq(start, stop) or seq (start, stop, by)* | Return the vector (start, start + step, start + 2 * step, ...)<br>The sequence stops when start + k * step is greater than end.<br>By: *increment of the sequence* |
| *rep(V,k)* | Return a vector which will be the concatenation of the vector V repeated k times |

**Applications :**

Shape Vectors (1,2,…,10) et (0.5,1,1.5,2,2.5).
What does the result of seq (2,6) give? and seq (1.2,6)?
What does the result of seq (10) give? and seq (10.2)?
Create the vector (1,2,4,7,1,2,4,7,1,2,4,7,1,2,4,7) with the function rep.
Create the vector (1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5) with the functions *rep* and *seq*.

V. Usual mathematical functions

| Functions | Descriptions |
|---|---|
| *sqrt(x)* | Return the root of x |
| *log(x)* | Return the natural logarithm of x |
| *log(x,n)* | Returns the logarithm of x to base n |
| *exp(x)* | Return the exponential of x |
| *sin(x)* | Return the sine of x |
| *cos(x)* | Returns the cosine of x |
| *tan(x)* | Returns the tangent of x |
| *asin(x)* | Return the arc sine of x |
| *acos(x)* | Return the arc cosine of x |
| *atan(x)* | Return the tangent arc of x |
| *factorial(n)* | Reference n! |
| *choose(n,p)* | Return the number of combinations of p elements among n |
| *floor(x)* | Return the integer part of x |
| *round(x)* | Returns the approximation of x to the nearest integer |
| *round(x,n)* | Returns the approximation of x to the nearest 10-n |

**Applications :**

Handle a few functions.

VI Boucles, conditions et fonctions

*1) if (condition) {instructions1} else {instructions2}*

After the if command, the “condition” block is tested, if this “condition” block is true then the “instructions1” command block is executed, if the “condition” block is false then the “instructions2” block is carried out.

If there is only one command in "instruction1", the braces surrounding "instructions1" are not required, so are the braces surrounding "instructions2". If there are at least two commands, then they are mandatory. The second part (else {instructions2}) is not mandatory if there is nothing to do if "condition" is false.

**Some examples:**

*if (pi > 3) Rep = TRUE else Rep = FALSE*
*Rep*

*a = 7 ; if (a %% 2 == 1) {a = a + 1 ; a = a / 2 ;} else {a = a + 2 ; a = a / 2 ;} ; a ;*

*if (cos(0) == 1) {print("cos(0) = 1 est une bonne réponse")}*

*if ((cos(0) == 1) & (sin(0) == 1)) {print("cos(0) = 1 et sin(0) = 1 sont des bonnes réponses") ; print("bravo") ;} else {print("au moins une mauvaise réponse se cache parmi les deux affirmations cos(0) = 1 et sin(0) = 1") ; print("Il faut réviser les fonctions trigonométriques");}*

**Note:** Under R, there is also an if else function (condition, a, b). This function returns *a* if "condition" is true, otherwise it returns *b*.

*2) while (condition) {instructions}*

The commands of the “instructions” block are executed as long as the “condition” block is TRUE. More precisely, R starts by testing the "condition" block:

- if it is TRUE (case 1) then all the “instructions” commands are executed.
- otherwise R passes to the next command without carrying out the “*instructions* ».

In case 1, once the commands of the “instructions” block have been carried out, it tests “condition” again and starts the same process over again: if the “condition” block is TRUE, then it performs the “instructions” commands, otherwise it passes to the next command without carrying out the "instructions" commands.

**Be careful, the loop only stopping if "condition" is true, the program can loop indefinitely! It is therefore important to be careful that the loop ends at one point or another.**

**Some examples:**

*i = 1 ; while (i <= 10) {i = i +1 ; print(i %% 2) ;}*

*i = 1 ; V=c(1,3,5,7,9) ; while (i <= length(V)) { i = i + 1 ; print(V[i-1]) ;}*

*a = 1 ; while (a != 1) {a = 2} ; a ;*

*i = 0 ; a = TRUE ; while (a) {i = 7 + i ; if (i %% 10 == 0) {a = FALSE}} ; i ;*

*i = 1 ; while (i <= 10) {print(i) ; i = i + 10 ; print(i)}*

*3) for (i in X) {instructions}*

The “for” command is, like while, a loop. However, its number of iterations is defined in advance by the length of the vector X. This command cannot loop indefinitely.

More precisely, R begins by giving the first value of the element X (if X is not empty in which case R passes to the next command) to i then R performs the commands in "instructions". It then gives the second value of the element X (if X has at least two elements) then R performs the commands in "instructions". It continues in the same way with a number of iterations equal to the length of the vector X.

**Some examples:**

```
for (i in 1:10) {a = i * (i + 1) ; print(a) ;}
for (i in c(1,3,5,7,9)) {a = i + 1 ; print(a) ;}
a = 1 ; for (i in c()) {a = 2} ; a ;
for (i in 1:10) {print(i) ; i = i + 10 ; print(i)}
```

What do you notice?

VII Functions

When we wish to execute at different times of a program the same series of instructions (depending or not on parameters), it is interesting to be able to group this series of commands once and for all under a generic name and to recall this series of commands instructions only with this name. This is the point of using the functions that R allows programming.

The definition of a function has the following structure:

```
ExFct = function (a = 2,b,c,...)
{
        instructions ;
        return(X)
}
```

Here the name given to the function is "ExFct". A function may require more than one parameter, above three are named "a", "b" and "c". The number of parameters is not limited as you may well have none. Even without a parameter in the function, it will be necessary to leave the parentheses, which is necessary for R to understand that it is a function. When defining the function, it is possible (not mandatory) to give default values as is the case for the "a" parameter to which we have assigned the value 2. In case we want to give a value x to the function, we do so using the return (x) statement.

When we want to call the function, we will do so by calling the name of the function and indicating, if necessary, the values of the parameters necessary for the execution of the function.

**Applications :**

```
EcritureBaseDix = function(a = 2, b = 3, c = 4)
{
        return(a * 100 + b * 10 + c) ;
```

```
}
```
What results for EcritureBaseDix (1,1,1)? For WriteBaseDix (1,1)? For WriteBaseTix (1)? For *EcritureBaseDix()* ? And for *EcritureBaseDix* ?

What do the Kesako and Goodberg functions do?

```
Kesako <- function(N = 100)
{
        i = 1
        C = c(2 : N)
        while (C[i] <= sqrt(N))
        {
                j = 2
                while (C[i] * j <= N)
                {
                        C = C[which(C != C[i] * j)]
                        j = j + 1
                }
                i = i + 1
        }
        return(C)
}
a = Kesako(100)

i = 1
Goodberg = function(N)
{
       while (i <= N / 2)
       {
                trouve = FALSE
                j = 1
                k = 1
                tmp = 0
                while ((j <= length(a)) & (!trouve))
                {
                        tmp = a[j] + a[k]
                        if (tmp == 2 * i) {trouve = TRUE} else
                        {
                         if ((tmp > 2 * i) | (k >= length(a)))
                         {
                                j = j + 1 ; k = j
                         } else
                         {
                                k = k + 1
                         }
                    }
                        if (trouve) print(c(2 * i, a[j], a[k]))
                }
                i = i + 1
```

```
        }
}
```

Display the first hundred terms of the Syracuse sequence defined by

$$\{U_{n+1} = \frac{U_n}{2} \; if \; U_n \; is \; even \; U_{n+1} = 3\,U_n + 1 \; if \; U_n \; is \; odd$$

You will send the suite to U_0 that you choose. Try with several values. What do you notice?

VIII Various

| Functions | Descriptions |
|---|---|
| *rm(x)* | Initialize the variable x |
| *ls(all = TRUE)* | List all the variables used since the start of the program |
| *scan(what = ... )* | Expects the user to enter data, by default numeric, or characters if we specify what = "character" |
| *rm(list=ls())* | Initializes all variables used |
| *# commentaires* | Placing a hash allows you to write comments after in a script or on the console |