

Introduction to **R** software

Kundan Singh
Research Scholar
Department of Mathematics
Indian Institute of Technology, Patna

16 August 2021

Chapter 5 : Function, Conditional Executions and Loops

Functions

syntax

```
Name <- function(Argument1, Argument2, ... )  
  {  
    expression  
  }
```

where `expression` is a single command or a group of commands

Function arguments with description and default values

- Function arguments can be given a meaningful name
- Function arguments can be set to default values
- Functions can have the special argument ‘...’

Functions (Single variable)

The sign `<-` is furthermore used for defining functions:

```
> abc <- function(x){  
     $x^2$   
}
```

```
> abc(3)  
[1] 9
```

```
> abc(6)  
[1] 36
```

```
> abc(-2)  
[1] 4
```

Functions (Multiple variable)

```
> abc <- function(x,y){  
     $x^2 + y^2$   
}
```

```
> abc(2,3)  
[1] 13
```

```
> abc(3,4)  
[1] 25
```

```
> abc(-2,-1)  
[1] 5
```

1. Conditional execution

Syntax

if (condition) {executed commands if condition is TRUE}

if (condition) {executed commands if condition is TRUE}

else { executed commands if condition is FALSE }

Please note:

- The condition in this control statement may not be vector valued and if so, only the first element of the vector is used.
- The condition may be a complex expression where the logical operators "and" (&&) and "or" (| |) can be used.

Example : 1

```
> x <- 5
```

```
> if ( x==3 ) { x <- x-1 } else { x <- 2*x }
```

Interpretation:

- If $x = 3$, then execute $x = x - 1$.
- If $x \neq 3$, then execute $x = 2*x$.

In this case, $x = 5$, so $x \neq 3$. Thus $x = 2*5$

Examples : 2

```
> x <- 3
```

```
> if ( x==3 ) { x <- x-1 } else { x <- 2*x }
```

Interpretation:

- If $x = 3$, then execute $x = x - 1$.
- If $x \neq 3$, then execute $x = 2*x$.

In this case, $x = 3$, so $x = 3-1$

2. Conditional execution

Syntax

`ifelse(test, yes, no)`

- Vector-valued evaluation of conditions .
- For the components in the vector-valued logical expression `test` which provide the value `TRUE`, the operations given by `yes` are executed.
- For the components in the vector-valued logical expression `test` which provide the value `FALSE`, the operations given by `no` are executed.

Example

```
> x <- 1:10  
> x  
[1] 1 2 3 4 5 6 7 8 9 10  
> ifelse( x < 6, x2, x + 1 )  
[1] 1 4 9 16 25 7 8 9 10 11
```

Interpretation

- If $x < 6$ (TRUE), then $x = x^2$ (YES) .
- If $x \geq 6$ (FALSE), then $x = x + 1$ (NO).
- So for $x = 1, 2, 3, 4, 5$, we get $x = x^2 = 1, 4, 9, 16, 25$
- For $x = 6, 7, 8, 9, 10$, we get $x = x + 1 = 7, 8, 9, 10, 11$

Control structures in R :

Loops

Repetitive commands are executed by loops

- for loop
- while loop
- repeat loop

1. The for loop

If the number of repetitions is known in advance (e.g. if all commands have to be executed for all cases $i = 1, 2, \dots, n$ in the data), a `for()` loop can be used.

Syntax

```
for (name in vector) {commands to be executed}
```

A variable with name `name` is sequentially set to all values, which contained in the vector `vector`.

All operations/commands are executed for all these values.

Example

```
> for ( i in 1:5 ) { print( i^2 ) }
```

```
[1] 1
```

```
[1] 4
```

```
[1] 9
```

```
[1] 16
```

```
[1] 25
```

Note: `print` is a function to print the argument.

2. The while() loop

If the number of loops is not known in before, e.g. when an iterative algorithm to maximize a likelihood function is used, one can use a `while()` loop.

Syntax

```
while(condition){ commands to be executed as long as condition is  
TRUE }
```

If the condition is not true before entering the loop, no commands within the loop are executed.

Example

```
> i <- 1  
> while (i<5)  
+ print(i^2)  
+ i <- i+2  
+  
[1] 1  
[1] 9
```

The programmer itself has to be careful that the counting variable `i` within the loop is incremented. Otherwise an infinite loop occurs.

3. The repeat loop

The repeat loop doesn't test any condition — in contrast to the `while()` loop — *before entering* the loop and also not during the execution of the loop.

Again, the programmer is responsible that the loop terminates after the appropriate number of iterations. For this the `break` command can be used.

Syntax

```
repeat{ commands to be executed }
```


Example:

```
> i <- 1  
> repeat{  
+ print( i^2 )  
+ i <- i+2  
+ if ( i > 10 ) break  
+}
```

```
[1] 1  
[1] 9  
[1] 25  
[1] 49  
[1] 81
```

Example:

Additionally, the command `next` is available, to return to the beginning of the loop (to return to the first command in the loop).

```
> i <- 1
> repeat{
+ i <- i+1
+ if (i < 10) next
+ print(i^2)
+ if (i >= 13) break
+}
[1] 100
[1] 121
[1] 144
[1] 169
```