Chapitre 4
La programmation avec R

1. Introduction

Précédemment, certains objets R ont été étudiés, à savoir : les vecteurs, les facteurs, les data frames, les matrices, etc. qui permettent le stockage en mémoire des données. Cependant, il existe des structures, essentielles à l'implémentation d'algorithmes, qui n'ont pas été abordées. Il s'agit d'outils à la base de la programmation, de la grammaire du langage, pour non seulement structurer mais aussi traduire en code le raisonnement du programmeur. Ainsi, ce chapitre portera sur les fondamentaux de la structuration du code avec R et traitera des structures de contrôle (les conditions et les boucles), des fonctions et de la programmation orientée objet (POO) notamment.

2. Les structures de contrôle

On distingue généralement et principalement deux catégories de structures de contrôle, celles qui servent à poser des conditions ou à tester une information d'une part et celles qui permettent la répétition d'une ou de plusieurs opérations d'autre part, que l'on appelle en programmation les boucles.

Mais avant d'y arriver, on va aborder un type de structure simple qui sert à grouper plusieurs instructions ayant ou non une suite logique.

2.1 Les structures de groupage d'instructions

2.1.1 Le point-virgule

Il permet d'écrire plusieurs instructions en une seul ligne de commande. Il est plus pratique de ne l'utiliser qu'en cas d'instructions silencieuses (notamment pendant l'initialisation de plusieurs objets) pour permettre une meilleure lisibilité du code. Ainsi, les instructions suivantes :

```
> # initialisation de plusieurs variables
> a <- 3
> b <- 5
> c <- b*b-a*c
> mu <- mean(1:12)
```

Peuvent être écrites comme ci-dessous :

```
> # initialisation de plusieurs variables
> a <- 3;b <- 5;c <- b * b - a * c ;mu <- mean(1:12)
```

2.1.2 Les accolades

R permet également de regrouper entre accolades une suite d'instructions pointant vers un résultat. L'exemple ci-dessous permet d'abord d'initialiser les variables **a** et **b** et de déterminer **c** :

```
> # détermination de c = a + b
> {
+    a <- 2.3
+    b <- 5.7
+    c <- a + b
+ }
> # ou une autre façon plus élégante
>  c <- {
+  a <- 2.3
+  b <- 5.7
+  a + b
+  }
> # afficher la valeur de c
> c
[1] 8
```

Remarque

L'utilisation d'accolades n'est vraiment pertinente que pour bien structurer la détermination de certains résultats impliquant plusieurs instructions intermédiaires, notamment lors de la construction de structures plus élaborées (les fonctions, etc.).

2.2 Les structures conditionnelles

2.2.1 La structure if...else

Pour les habitués de Microsoft Excel, la structure si…alors…sinon est équivalente à la fonction **SI()** de ce fameux tableur, avec R cette fonction prend plusieurs formes :

```
# la première forme if ... le si simple
if(" Poser une condition ici "){
" Placer Les instructions ici si vrai "
}
# la seconde forme if …else… le si …sinon
if( " Poser une condition ici "){
  " Placer les instructions ici si vrai "
}else{
 " Placer les instructions ici sinon "
}
# la troisième forme la fonction ifelse() pour un test vectoriel
ifelse(" Poser une condition sur un vecteur ici ",
"si vrai instruction ",
" sinon instructions ")
```

Le principe de fonctionnement est simple, il suffit de poser un test ou une condition, c'est-à-dire une opération à même de générer une valeur booléenne à savoir **TRUE** ou **FALSE**, ensuite placer à la suite une ou plusieurs instructions selon que le test est vrai ou faux. Ainsi, une instruction ne sera exécutée que si la valeur booléenne générée par le test est vrai (**TRUE**).

```
> # si vrai afficher le texte
> if(TRUE){
+   print(" Hey !!! ça fonctionne ! ")
+ }
[1] " Hey !!! ça fonctionne ! "
> # si faux rien ne s'affiche
> if(FALSE){
+   print("Hey !!! ça ne fonctionne pas ! ")
+ }
> # avec un test générant un booléen FALSE
> if(5<3){
+   print("rien ne s'affiche")
+ }
> # avec un test générant un booléen TRUE
> if(5>3){
+   print("5 est bien supérieur à 3")
+ }
[1] "5 est bien supérieur à 3"
```

Mis à part des opérations logiques comme ci-dessus, il s'agit le plus souvent de tester la structure d'un objet notamment pour en déterminer la classe afin d'effectuer une suite d'instructions précises selon la classe. Les fonctions R permettant de réaliser de telles conditions sont par convention nommée selon le modèle **is.nom_de_la_classe()**. Les plus fréquemment utilisées sont **is.vector()**, **is.numeric()**, **is.string()**, etc. Pour obtenir la liste complète de ce type de fonctions très utiles lors de l'utilisation des structures conditionnelles, il faut procéder comme ceci :

```
> ls(pattern = "^is", baseenv())
 [1] "is.array"
 [2] "is.atomic"
 [3] "is.call"
 [4] "is.character"
 [5] "is.complex"
 [6] "is.data.frame"
 [7] "is.double"
 [8] "is.element"
 [9] "is.environment"
[10] "is.expression"
[11] "is.factor"
[12] "is.finite"
[13] "is.function"
[14] "is.infinite"
[15] "is.integer"
[16] "is.language"
[17] "is.list"
[18] "is.loaded"
```

En voici une illustration avec la fonction **is.numeric()** qui permet de tester si un objet est de classe numérique :

```
> v <- 100.0
> # tester si v est numérique
> if (is.numeric(v)) {
+    # si vrai en déterminer le logarithme
+    log10(v)
+ }
[1] 2
>
```

Toutefois, l'utilisation la plus répandue reste la seconde forme, **if…else** (si sinon), qui se prête plus à l'algorithmique :

```
> # exemple affichant la valeur absolue d'un nombre
> # aléatoirement généré compris entre -10 et 10
> a <- runif(1, -10, 10)
> if (a > 0) {
+    print(a)
+ } else {
+    print(-a)
+ }
[1] 5.802002
>
```

Sa forme la plus complexe consiste à imbriquer plusieurs structures **if…else**, comme l'illustre le code ci-dessous :

```
> # demander à l'utilisateur de saisir une valeur
> x <- scan(nmax = 1)
1:
7

Read 1 item
> ## tester si x est bien un nombre
> if (is.numeric(x)) {
+    msg <- paste(x, ", est un nombre")
+    ## structure if...else imbriquée
+    # testant la parité de x
+    if (x %% 2 == 0) {
+    cat(msg, "pair")
+    } else {
+    cat(msg,"impair")
+    }
+    # fin du test de parité
+ } else {
+   cat(x,"n'est pas un nombre")
+ } # fin du test du type de x
7 , est un nombre impair
```

La troisième forme, **ifelse**, est une fonction qui opère un test sur un vecteur de données contrairement aux formes précédentes qui ne testent que des scalaires :

```
> x <- c(-5, 7, 0, 9, -2)
> # valeur absolue du vecteur x
> ifelse(x >= 0, x, - x)
[1] 5 7 0 9 2
> # logarithme de x si possible
> ifelse(x > 0,
+       paste0("log(",x,") vaut ",log(x)),
+       paste0("log(",x,") n'est pas possible"))
[1] "log(-5) n'est pas possible"
[2] "log(7) vaut 1.94591014905531"
[3] "log(0) n'est pas possible"
[4] "log(9) vaut 2.19722457733622"
[5] "log(-2) n'est pas possible"
```

Naturellement, cette dernière forme accepte également l'imbrication comme les autres formes de condition, mais ne reste pertinente que pour tester des vecteurs.

2.2.2 La structure switch()

Contrairement aux autres structures conditionnelles, cette structure est un peu particulière. R et les autres langages de programmation courants présentent plutôt la structure **switch()** comme un menu avec des choix ou sélections multiples de type clé-valeur qui prend obligatoirement en argument une clé et vérifie puis renvoie la valeur sur laquelle pointe la clé :

```
switch("Clé de sélection ",
       "sélection 1",
       "sélection 2",
       ".........",
       "sélection n",
       "sélection par dé faut ")
```

La clé de sélection est nécessaire à cette structure, elle permet d'indexer une des options ou sélections prévues. Elle peut être une chaîne de caractères ou numérique :

```
> # clé en charactère
> switch("b",
+     a = c(2.0,3.6,9,0,3),
+     b = "Bonjour le monde",
+     c = data.frame(1:4,LETTERS[1:4]),
+     "option par defaut")
[1] "Bonjour le monde"
> # clé numerique et affectation
> val <- switch(2,
+       a = c(2.0, 3.6, 9, 0, 3),
+       b = "Bonjour le monde",
+       c = data.frame(1:4, LETTERS[1:4]),
+       "option par defaut")
> val # afficher  val
[1] "Bonjour le monde"
```

Toutefois, en l'absence de correspondance c'est soit l'objet **NULL**, soit l'option ou sélection par défaut qui est retourné si cette dernière est prévue :

```
> # cas de clé inexistante
> val <- switch("d",
+       a = c(2.0, 3.6, 9, 0, 3),
+       b = "Bonjour le monde",
+       c = data.frame(1:4, LETTERS[1:4]),
+       "option par defaut")
> val
[1] "option par defaut"
```