Introduction à la programmation sous R

L. BELLANGER

lise.bellanger@univ-nantes.fr

Page web: http://www.math.sciences.univ-nantes.fr/~bellanger/

Les opérateurs logiques sous R

Les opérateurs logiques en R sont :

- ET : **&**
- OU : **|**
- NON : **!**

Les opérateurs logiques sous R

Exemple :

ET &

```
> TRUE & FALSE
# [1] FALSE
> x <- 5 x > 2 & x < 10
# [1] TRUE
```

OU |

```
> TRUE | FALSE
# [1] TRUE
> x <- 5 x <= 10 | x > 6
# [1] TRUE
```

NON !

```
> !TRUE #
[1] FALSE
> x <- 5 !(x == 5)
# [1] FALSE
```

Dans le cas de vecteurs

```
> c(TRUE, TRUE, FALSE) & c(TRUE, FALSE, FALSE)
# [1] TRUE FALSE FALSE
> c(TRUE, TRUE, FALSE) | c(TRUE, FALSE, FALSE)
# [1] TRUE TRUE FALSE
> !c(TRUE, FALSE)
# [1] FALSE TRUE
```


La structure « if …else »

- If

```
if (condition) { # TRUE
        expression
}
```

Exemple :

```
> x <- -2
> if (x < 0) {
      print("x est un nombre négatif")
}
# [1] "x est un nombre négatif"
```

La structure « if …else »

- If … else

```
if (condition) { # TRUE
        expression1
} else {
        expression2
}
```

Exemple :

```
> x <- -2
> if (x < 0) {
       print("x est un nombre négatif")
} else {
       print("x est un nombre positif ou zéro")
}
# [1] "x est un nombre négatif"
```


La structure « if …else »

- If … else if …. else

Exemple :

```
> x <- 6

> if (x < 0) {
       print("x est un nombre négatif")
} else if (x == 0) {
       print("x vaut zéro")
 } else {
       print("x est un nombre positif")
 }
# [1] "x est un nombre positif"
```


Les boucles

Deux types de boucles en R :
- Boucle « while »
- Boucle « for »

La boucle « while »

Boucle "while" : similaire à une condition "if" répétée.

Structure :

```
while (condition) {
       expression
}
```

Exemple :

```
> compteur <- 1
> while (compteur <= 4) {
       print(paste("Le compteur vaut", compteur))
       compteur <- compteur + 1
}
# [1] "Le compteur vaut 1"
# [1] "Le compteur vaut 2"
# [1] "Le compteur vaut 3"
# [1] "Le compteur vaut 4"
Compteur
# [1] 5
```

ATTENTION *: ne pas oublier d'incrémenter le compteur, sinon boucle infinie !!!*
Pour arrêter une boucle infinie : ctrl + C (sous Windows) ou bien le bouton "stop" de RStudio.

La boucle « repeat »

Boucle "repeat" : quasi équivalente à la boucle « while », sauf que la condition d'arrêt n'est pas placée au même niveau.

Boucle « while » :

```
> compteur <- 1
> while (compteur <= 4) {
          print(paste("Le compteur vaut", compteur))
          compteur <- compteur + 1
}
# [1] "Le compteur vaut 1"
# [1] "Le compteur vaut 2"
# [1] "Le compteur vaut 3"
# [1] "Le compteur vaut 4"
compteur
# [1] 5
```

Boucle « repeat » :

```
> compteur <- 1
> repeat{
          print(paste("Le compteur vaut", compteur))
          compteur <- compteur + 1
          if(compteur >4 ) {
          break
          }
}
# [1] "Le compteur vaut 1"
# [1] "Le compteur vaut 2"
# [1] "Le compteur vaut 3"
# [1] "Le compteur vaut 4"
compteur
# [1] 5
```


La boucle « for »

Structure :

```
for (variable in sequence) {
       expression
}
```

Exemples :

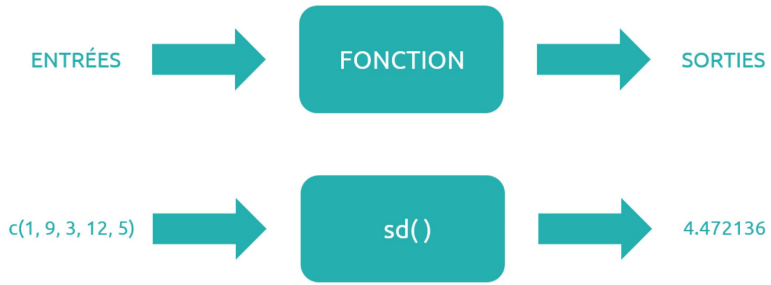
```
> for (compteur in 1:4) {
       print(paste("Le compteur vaut", compteur))
}
# [1] "Le compteur vaut 1"
# [1] "Le compteur vaut 2"
# [1] "Le compteur vaut 3"
# [1] "Le compteur vaut 4"
```


Les fonctions

=> Concept très important et très puissant en R.

On en a déjà vues sans y prêter attention :

```
data.frame(); list() ; print() ; plot() ; summary(); ..;
```



Les fonctions

Appeler une fonction dans R

```
> sd(c(1,9,3,12,5))
# [1] 4.472136
#ou
> valeurs <- c(1,9,3,12,5)
> sd(valeurs)
# [1] 4.472136
#ou
> valeurs <- c(1,9,3,12,5)
> ecart_type<-sd(valeur)
> ecart_type
# [1] 4.472136
```

Les fonctions

Documentation d'une fonction

```
> ?sd
#ou
> help(sd)
```

Plusieurs paramètres possibles pour la fonction `sd` …

```
> args(sd)
# function (x, na.rm = FALSE)
# NULL
```

Exemple :

```
> valeurs <- c(1, 9, 3, 12, 5, NA)
> sd(valeurs)
# [1] NA
> sd(valeurs, na.rm = TRUE)
# [1] 4.472136
> sd(valeurs, TRUE) # accès par position
# [1] 4.472136
> sd(x = valeurs, na.rm = TRUE) # accès par nom
# [1] 4.472136
```


Les fonctions

Ecrire une fonction

```
> ma_function <- function(args1, args2) {
      expressions
}
```

=> Sert à résoudre un problème ou effectuer des actions répétitives.

Exemple 1 :

```
> carre <- function(x) {
      x*x
}
> ls()
# [1] "carre"
> carre(2)
# [1] 4
```

Remarque : *ATTENTION, les variables définies au sein d'une fonction (tq "x" dans l'exemple ci-dessous) ne seront pas accessibles en dehors de cette fonction.*

```
> carre(x=5)
# [1] 25
> x
# Error in eval(expr, envir, enclos): objet 'x' introuvable
```


Les fonctions

L'utilisation de "return"

```
> carre <- function(x) {
      y <- x*x
      return(y)
}

> carre(2)
# [1] 4

> res <- carre(3); res
# [1] 9
```

Remarque : Utiliser la fonction return si on doit retourner un résultat sans être à la dernière ligne de la fonction (à l'intérieur d'un bloc conditionnel, par exemple.

L'utilisation de return *à la toute fin d'une fonction est tout à fait inutile et considérée comme du mauvais style en R.*

Les fonctions

Ecrire une fonction

Exemple 2 :

```
> diviser <- function(a,b) {
      a/b
}
> diviser(10,2)
# [1] 5
> diviser(10)
# Error in diviser(10): l'argument « b » est manquant …
```

Fixer une valeur par défaut

```
> diviser <- function(a,b=1) {
      a/b
}
> diviser(10)
# [1] 10
```

Les fonctions

Ecrire une function

Exemple 2 :

```
> diviser(10, 0)
# [1] Inf
```

Prévoir un cas d'exception

```
> diviser <- function(a, b = 1) {
         if (b == 0) { return("Division par zéro interdite")
        # va retourner le message d'erreur et quitter la fonction
        } else { return(a / b)
        }
 }
> diviser(10, 0)
# [1] "Division par zéro interdite"
```


Les fonctions

Ecrire une function

Exemple 3 : Une fonction sans argument

```
> lancer_de <- function() {
        face <- sample(1:6, size=1) # echantillon aleatoire
        face
 }
> lancer_de()
# [1] 1

> lancer_de()
# [1] 4
```


Les fonctions

Remarques :

- *Lorsque l'on a chargé un package dans R, l'ensemble des fonctions qui le composent sont disponibles et directement utilisables en les appelant par leur nom.*
- *Avant d'écrire une fonction, pensez à faire une rapide recherche pour voir si une telle fonction n'existe pas déjà dans un des nombreux packages R disponibles.*

Les fonctions

Ecrire une function

Exemple 4 : plus complexe

```
> passage <- function(note) {
        if (note >= 10) {
        print("Matière validée")
        return(note)
        } else {
          print("Matière à repasser")
         return(0)
        }
 }

> carnet_notes <- c(12, 8, 15, 20, 4)
> passage(carnet_notes[1])
# [1] "Matière validée "
# [1] 12
> passage(carnet_notes[2])
# [1] "Matière à repasser"
# [1] 0
```

Les fonctions

Ecrire une function

Exemple 4 : plus complexe

```
> passage_complet <- function(carnet, return_moy=TRUE) {
        if (return_moy) {
          notes <- NULL
          for (n in carnet) {
                notes <- c(notes, passage(n))
          }
          return(mean(notes))
        }
        else {
          return(NULL)
         }
 }
> carnet_notes <- c(12, 8, 15, 20, 4);passage_complet(carnet_notes)
# [1] "Matière validée«
# [1] "Matière à repasser"
# [1] "Matière validée"
# [1] "Matière validée"
# [1] "Matière à repasser"
# [1] 9.4
```


Références

Sur R:

- https://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.pdf : Introduction à la programmation en R, Vincent Goulet