# *Les bases de la programmation en R*

Créé par Sébastien Durand
24/05/06
Révision D. Borcard 07/09/06

Bio2041
Département de sciences biologiques
Université de Montréal

### *Introduction aux commandes de base*

R est en fait un langage de programmation. Les commandes qui suivent sont généralement utilisées à l'intérieur des fonctions. De prime abord, ces fonctions peuvent vous sembler nombreuses, mais sachez qu'elles sont très faciles à utiliser et à mémoriser. Dans le langage R, nous disposons des structures de contrôle généralement retrouvées dans tous les langages de programmation. Voici une brève description, qui j'espère, démystifiera les bases de la programmation. Dans ce document, nous abordons les boucles, les conditionnelles et la récursivité.

Toutes les fonctions mentionnées ci-dessous, à l'exception de « else », « break » et « next », **utilisent une ou des conditions.** Ces dernières sont inscrites entre les parenthèses qui suivent la fonction. Une condition est une situation particulière qui doit être rencontrée. Une fois cette condition vérifiée, si le résultat est **vrai**, le code inscrit entre les accolades « {…} » est **exécuté**. Cependant si le résultat de la condition est **faux**, le code entre les accolades est simplement **ignoré**. Afin d'écrire ces conditions correctement, consultez les exemples décrivant l'utilisation que nous faisons des **opérateurs de comparaison et logiques** dans le document GuideMacTextWranglerR.pdf.

### *Les données de départ*

Afin d'exercer tous les exemples décrits dans ce document, exécutez dans votre console R les lignes suivantes :

```
# Voici les données que nous allons utiliser au long de ce document

# Le rayon moyen de cinq types de cellules bactériennes
TailleCell <- c(10, 18, 5, 64, 25)
# Nombre de cellules disponibles au départ
NbCell <- c(2, 10, 4, 7, 13)
# Rythme de division cellulaire (minutes)
DivCell <- c(20, 17, 18, 21, 19)
# Crée un tableau à l'aide de ces trois vecteurs
TabCell <- rbind(TailleCell, NbCell, DivCell)
# Donne des noms de colonnes au tableau
colnames(TabCell) <- c("A", "B", "C", "D", "E")
```

### *Méthodes de travail*

Lorsque vous travaillerez avec des fonctions conditionnelles, probablement imbriquées dans votre fonction principale, vous ferez face à des situations qui peuvent être déconcertantes où plusieurs paires d'accolades se suivent et où d'autres sont imbriquées. Voici quelques conseils qui vont améliorer la lisibilité de votre code et aussi diminuer les risques d'erreurs associées à l'utilisation de multiples parenthèses, crochets, et accolades.

- **Assurez- vous de toujours fermer l'accolade que vous ouvrez**, ensuite déplacez votre curseur entre les accolades. Il en va de même pour les crochets, les parenthèses et les guillemets. De cette façon, vous vous assurez de ne pas en oublier une seule.
- Utilisez un ou plusieurs tabulateurs au début de vos lignes de commandes afin de hiérarchiser visuellement le contenu de votre fonction principale.

***Les conditionnelles***

Dans ce groupe de fonctions, on retrouve trois commandes ; le « **`if`** », le « **`else`** » et le « **`ifelse`** ». Ces commandes sont extrêmement utiles, car elles permettent de poser des conditions simples. Elles correspondent au « **si** » et au « **sinon** » de la langue française. Voici des exemples.

## *Le « if »*

```
if( condition ) {
      …
}
```

- Le « `if` » est généralement suivi d'un « `else` », cependant ce dernier n'est pas obligatoire.
- Le « `if` » peut être suivi par une série d'autres « if » utilisant des conditions complémentaires.

Exemple 1: Si la taille des cellules est plus grande ou égale à 25, on affiche un commentaire.

```
# Teste la quatrième colonne de notre tableau
i <- 4
# Lance la commande if sur la ligne 1 et colonne 4 de notre tableau
if( TabCell[1, i] >= 25 ) {
      # Affiche le commentaire
      cat("Grosse cellule", "\n")
}
```

Maintenant, retapez la même commande (flèche vers le haut) et remplacez la valeur de « `i` » par une autre, vous verrez si un résultat est affiché ou non.

*Le « else »*

```
else{
      …
}
```

- À la différence du « `if` », le « `else` » n'a pas de condition. Il englobe simplement ce que le « `if` » a exclu.
- Notez qu'il ne peut pas y avoir plus de « `else` » que de « `if` », car cette fonction dépend du nombre de « `if` ».

Exemple 2 : Ajoutons à l'exemple précédent un « `else` ».

```
# Teste la quatrième colonne de notre tableau
i <- 4
# Lance la commande if sur la ligne 1 et colonne 4 de notre tableau
if( TabCell[1, i] >= 25 ) {
      # Affiche le commentaire
      cat("Grosse cellule", "\n")
} else {
      # Affiche le commentaire
      cat("Petite cellule", "\n")
}
```

Maintenant, retapez la même commande (flèche vers le haut) et remplacez la valeur de « `i` » par une autre, vous verrez si un résultat différent est affiché ou non.

*Le « ifelse»*

```
ifelse( condition, Vrai, Faux )
```

- Le « `ifelse` » combine dans une même commande le « `if` » et le « `else` ».
- Son utilisation est souhaitable pour des situations simples.
- Tapez ?ifelse et essayez l'exemple
- Si le résultat de la condition est vrai, la fonction exécute les commandes qui remplacent le mot « Vrai ».
- Si le résultat de la condition est faux, la fonction exécute les commandes qui remplacent le mot « Faux »

Exemple 3 : Reprenons l'exemple précédent, mais à l'aide d'une fonction « `ifelse` »

```
# Teste la quatrième colonne de notre tableau
i <- 4
# Stocke le résultat du ifelse dans res
ifelse( TabCell[1, i] >= 25, res <- "Grosse cellule",
res <- "Petite cellule" )
cat(res, "\n")                    # Affiche le res
```

***Les boucles***

La boucle est un mécanisme clef de programmation. Il permet à l'usager de sauver du temps en répétant le code contenu entre les parenthèses qui suivent la boucle.

## *La boucle « for »*

```
for( compteur in sequence )
```

- Le « `compteur` » est initialisé dans cette fonction. Dans ce cas, il prend, au premier passage, la première valeur de l'objet « `sequence` ». Dans le deuxième passage, le compteur prend la deuxième et … jusqu'à ce que tous les éléments trouvés dans le vecteur « sequence » soient passés dans la boucle.
- On utilise cette boucle lorsqu'on connaît à l'avance la séquence ou la série de valeurs que devra prendre le « `compteur` ».

Exemple 4 : Nous allons créer une petite boucle simple pour débuter.

```
# Crée une boucle qui met x au carré 5 fois
x <- 2                        # Donne la valeur initiale de x
for( i in 1 : 5 ){            # Crée la boucle
      x<- x^2                 # Met la valeur au carré
      cat(x,"\t")             # Affiche la valeur de x
}
```

Exemple 5: Nous allons créer une petite boucle qui utilise la fonction « `ifelse` ».

```
# Trouve le nombre de colonnes dans notre tableau
NbCol <- ncol( TabCell )
for( i in 1 : NbCol ) {
      # Stocke le résultat du ifelse dans res
      res <- ifelse( TabCell[1, i] >= 25, "Grosse cellule",
      "Petite cellule" )
      # Trouve le nom de la colonne
      NomCol <- colnames( TabCell )
      # Affiche le nom de colonne approprié et le qualificatif de res
      cat(NomCol[i], "est une", res, "\n")
}
```

*La boucle « while »*

```
while( condition ){
      …
}
```

- Cette boucle correspond au mot « tant que » : tant que la condition est **vraie,** la boucle est exécutée.
- On utilise cette boucle lorsqu**'on ne sait pas combien de boucles** nous devons effectuer.
- L'élément inscrit dans la condition de cette boucle doit déjà exister.
- Cette boucle requiert la gestion du compteur

Exemple 6 : Nous allons créer une petite boucle qui utilise la fonction « `while` ».

```
i <- 1                  # Initialise le compteur
while( i <=10 ) {
      cat(i, "\n")      # Affiche la valeur de i
      i <- i+2          # Augmente la valeur de i
}
```

Exemple 7 : Combien de cycles de division cellulaire me faut-il pour avoir 1000000 cellules à partir d'un nombre "Nb" de cellules de départ.

```
i <- 1                  # Choisit la colonne du tableau de données
j <- 0                  # Crée un compteur qui va compter les boucles
Nb <- TabCell[2,i]      # Choisit la valeur de départ dans la col. i
while( Nb <= 1000000 ) {           # Début de la boucle
      Nb <-Nb^2               # Chaque cellule se divise
      j <- j+1                # Augmente de un la valeur du compteur
}
NomCol<- colnames(TabCell)    # Trouve les noms de colonnes
# Affiche le résultat
cat("Afin d'arriver à 1000000 de cellule du groupe", NomCol[i],
"il faut", TabCell[3,i]*j, "minutes ", "\n")
```

*Deux fonctions à connaître : « break » et « next »*

Ces deux fonctions ne peuvent être utilisées qu'à l'intérieur d'une boucle. Advenant le cas où vous souhaiteriez arrêter une boucle dans une situation particulière, il faut utiliser la fonction « `break` ». La boucle se terminera et le code qui suit la boucle sera exécuté.

Exemple 8 : Il y a trop de cellules si leur nombre est supérieur ou égal à 10.

```
NbCol <- ncol(TabCell)              # Nb de colonnes dans le tableau
for( i in 1 : NbCol) {              # Crée la boucle
      if( TabCell[2,i] >= 10 ) {    # Si i plus grand ou égal à 10
            cat("Il y a trop de cellules à la colonne", i,"\n")
            break                   # Force l'arrêt de la boucle
      }                             # Fin du if
      cat("Il y a", TabCell[2,i], "cellules à la colonne", i, "\n")
}
```

La fonction « `next` » ne fait que sauter une boucle. Donc, une fois la fonction « `next` » exécutée, le code reprend au début de la boucle.

Exemple 9 : Il y a trop de cellules si leur nombre est supérieur ou égal à 10. Nous devons vérifier toutes les colonnes.

```
NbCol <- ncol(TabCell)              # Nb de colonnes dans le tableau
for( i in 1 : NbCol) {              # Crée la boucle
      if( TabCell[2,i] >= 10 ) {    # Si i plus grand ou égal à 10
            cat("Il y a trop de cellules à la colonne", i,"\n")
            next                    # Saute au début de la boucle
      }                             # Fin du if
      cat("Il y a", TabCell[2,i], "cellules à la colonne", i, "\n")
 }
```

Remarquez que le « cat » suivant l'accolade fermant le « if » n'est pas affiché pour les colonnes 2 et 5.

*La boucle « repeat »*

```
repeat {
      ...
if(...) { break }
      ...
}
```

- Cette boucle **répète à l'infini** les commandes qui lui sont données entre ses accolades. Afin de s'assurer que cette fonction s'arrête, **nous devons utiliser la fonction « break ».**

Exemple 10 : Affiche et augmente la valeur de i à chaque boucle. Brise la boucle lorsque i est égal à 5.

```
i <- 0
repeat{
      i <- i + 1
      cat(i, "\n")            # Affiche la valeur de i
      if( i == 5 ) { break }  # Tout le if est exécuté sur une ligne
}
cat("Nous nous sommes arrêtés à",i , "\n")
```

***Comment sortir d'une boucle sans fin***

Si votre boucle ne semble pas vouloir s'arrêter, appuyer sur l'icône « `Stop` » qui se trouve au haut de votre console. Il ne vous restera plus qu'à vérifier ce qui cloche dans votre fonction.