Initiation à la programmation sous R

Christophe Genolini

23 novembre 2006

Table des matières

1 Les variables

Les *variables informatiques* (à distinguer des variables statistiques) servent à stocker les informations que l'on traite, aussi bien nos données de base que les résultats.

1.1 Trois types unitaire

1.1.1 Les nombres

```
> A <- 5
> A
```

1.1.2 Les booléens

Ils peuvent prendre les valeurs True et False

```
> B <- TRUE
> B
```

1.1.3 Les alphanumériques

Ils contiennent les chaînes de caractères

```
> D <- "bonjour"
> B1<-"H" ; B2<-"F"
```

1.2 Manipulation d'objet unitaire

1.2.1 Généralités

permet d'écrire des commentaires. **R** ignore ce qui suit le #.

```
> A + 2 # Cette phrase ne change rien, R ne la lit pas.
```

R fait la distinction entre les majuscules et les minuscules. `> a-1` génèrera une erreur.

1.2.2 Opérations arithmétique de base

```
> A+3    # Addition
> A*2    # Multiplication
> A^2    # Puissance
> A/3    # Division par un réel
> 1/A    # Inverse
> log(A) # Logarithme
> cos(A) # Cosinus
```

1.2.3 Affichage

```
> cat(A) # Affiche la valeur de la variable A
> cat("A") # Affiche la chaîne de caractère "A"
> cat("\n") # Saute une ligne
> cat(A,"A") # Affiche la valeur de A, puis la chaîne de caractère "A"
> cat("D=",D,"\nB=",B,"\n") # Que fait cette instruction?
```

1.3 Vecteurs et Matrices

Un vecteur est une suite de plusieurs objets unitaires, tous du même type élémentaire. Pour les construire, on peut utiliser la fonction `c()` , concaténatrice :

```
> AA<-c(1,2,10,4,1,6) # Crée le vecteur (1,2,10,4,1,6)
> BB <- 1 :5 # Crée le vecteur (1,2,3,4,5)
> BB <- 5 :1 # Crée le vecteur (5,4,3,2,1)
```

On peut ensuite manipuler un vecteur pratiquement comme un type élémentaire. En particulier, les opérations arithmétiques de base sont appliquées a chacun des éléments du vecteur :

```
> AA+1
> AA*2
> AAA <- c(AA,AA+5,AA*2,AA*AA)
> AAA
> B<-c(B1,B2,B1,B1,B2,"Neutre")
```

Pour accéder a un élément particulier du vecteur, on écrit le nom du vecteur suivit du numéro de l'élément entre `[` et `]` . On peut également sélectionner plusieurs éléments grâce à la fonction concaténatrice. Il est même possible de sélectionner plusieurs fois le même élément :

```
> AAA[3] # Donne le troisième élément de AAA
> AAA[c(3,5)] # Donne le troisième et le cinquième éléments de AAA
> AAA[c(3,5,3)] # Donne le troisième, le cinquième puis le troisième éléments de AAA
```

Une matrice est un tableau constitué de plusieurs objets unitaires, tous du même type élémentaire. Pour les construire, on utilise un vecteur a qui on attribue des dimensions :

```
> dim(B) <- c(3,2) # Transforme le vecteur B en une matrice a 3 lignes et 2 colonnes
```

Là encore, les opérations arithmétiques affectent chaque élément.
Pour accéder a un élément particulier de la matrice, on écrit le nom de la matrice suivit du numéro de ligne puis de colonne. On peut également accéder a une ligne entière en ne spécifiant que le numéro de ligne (et rien la ou devrait se trouver le numéro de colonne) et symétriquement, on peut accéder à une colonne en ne spécifiant que le numéro de colonne.

```
> B[1,2] # Donne l'élément situé ligne 1 et colonne 2
> B[1,] # Donne la première ligne
> B[,2] # Donne la deuxième colonne
> B[c(1,3,1),2] # Donne le premier puis troisième puis premier élément de la colonne
2.
```

A noter qu'un vecteur n'a pas de dimension. En particulier, un vecteur a qui l'on attribue les dimensions (6,1) n'est plus considéré comme un vecteur mais comme une matrice à une colonne.

1.3.1 data.frame

Un data.frame est une suite de vecteur qui peuvent être de type différent. Un data.frame est l'objet idéal pour stocker les bases de données du statisticien.

```
> A<-c(21,20,23,19,21,26)
> B<-c("h","f","h","h","f","Neutre")
> Dn <- data.frame(Age=A,Genre=B)
> Dn
> dim(Dn)
```

Un data.frame donne généralement des noms à ces colonnes. On peut également spécifier des noms pour les lignes.

```
> colnames(Dn) # Affiche le nom des colonnes
> colnames(Dn) <- c("Age","Sexe") # Modifie le nom des colonnes
> row.names(Dn) # Affiche le nom des lignes
> row.names(Dn) <- c("A","E","R","T","Y","U") # Devinez ?
```

On peut accéder à un élément particulier d'un data.frame de la même manière que pour une matrice : `Dn[3,2]` donne l'élément situé ligne 3 et colonne 2. On peut également spécifier le nom exact de la colonne en le faisant précéder du symbole `$` . On peut également mélanger les deux méthodes :

```
> Dn$Age # Donne la colonne Age
> Dn[,1] # Donne la colonne Age
> Dn$Age[3] # Donne la troisième ligne de la colonne Age
```

Enfin, il est possible de sélectionner uniquement les lignes correspondant a un certain critère en le spécifiant dans les accolades, la ou sont spécifiées les lignes :

```
> Dn[Dn$Sexe=="H",1] # Sélectionne les lignes pour lesquelles la variable sexe vaut
"H", puis affiche la colonne 1 des lignes ainsi sélectionnées.
> Dn[Dn[,2]=="H",1] # Même chose
> Dn[Dn[,2]=="H",]$Age # Même chose
```

1.4 Exercice

Construire un data.frame qui contient le Prénom, l'age et la réussite au bac (codé en booléen) de 5 individus

1.5 Quelques erreurs à ne pas faire...

- **R** considére qu'un booléen a FALSE vaut 0 et qu'un booléen a TRUE vaut 1. On peut donc écrire `TRUE + 2` . En toute logique, ça n'a aucun sens (TRUE OU TRUE, cela a du sens, 2+3 cela a aussi du sens, mais TRUE + 2 cela n'a pas de sens), c'est comme additionner des pommes et des poires. Cela s'appelle faire du polymorphisme, c'est la source de nombreuses erreurs. INTERDIT
- La fonction `attach()` est également une source de confusion. INTERDIT.

2 Fonctions

Une fonction est une instruction définit par **R**. Elle exécute une certaine tache, tache qui dépend d'argument qu'on lui donne. Les fonctions sont toujours suivit de `(liste d'argument)` . S'il n'y a aucun argument a fournir, la fonction est suivit de `()`

```
> ls() # donne la liste des variables présente en mémoire
> summary(Dn) # Résume l'analyse univarié de toutes les colonnes de Dn
> boxplot(Dn$Age) # Trace la boite a moustache de la variable Dn$Age
```

Grande force des logiciels de programmation, l'utilisateur peut définir ses propres fonctions :

```
> surfaceRectangle <- function(longeur,largeur){
>   result <- longeur * largeur
>   return(result)
> }
```

Ensuite, vous pourrez calculez une surface a chaque fois que vous en aurez besoin simplement en utilisant votre function `surface` :

```
> surfaceRectangle(3,4)
```

Une fonction peut faire appelle a une autre fonction :

```
> volumePave <- function(longeurP,largeurP,hauteurP){
>   result <- hauteurP * surfaceRectangle(longeurP,largeurP)
>   return(result)
> }
```

2.1 Exercice

Construire une fonction qui calcule le nombre de combinaison possible en tirant p objets parmi n (rappel : $C_n^p = \frac{n!}{p!(n-p)!}$

3 Branchement

Les branchements constituent un des 4 piliers de la programmation. Un branchement permet de réaliser soit l'action 1, soit l'action 2. Cela dépendra d'une condition. La syntaxe générale d'un branchement (syntaxe pour n'importe quel langage) est donc :

`Si <condition est vrai> Alors <exécute action 1> Sinon <exécute action 2>`

Sous **R**, la syntaxe est `if(<condition>){<action 1>}else{<action 2>}` :

```
> age <- 17
> if(age <18){
>   cat("Tu as ",age," ans, tu es mineur")
> }else{
>   cat("Tu as ",age," ans, tu es majeur")
> }
```

ATTENTION : Il peut arriver que l'action 2 soit de ne rien faire. La tentation est forte d'écrire une formule simplifiée `if(age <18){cat("Tu as ",age," ans, tu es mineur")}` . Cette simplification est fortement déconseillée, elle est la source de nombreuses erreurs.
Naturellement, les actions peuvent être beaucoup plus complexes, être constitué de plusieurs instructions ou contenir elle-même une condition.

3.1 Exercice

Écrire une fonction qui prend pour argument un age et qui affiche "mineur" si l'age est inférieur a 18 ans, "sénior" s'il est supérieur a 60 ans et "plein tarif" dans les autres cas.

3.2 Prédicat logique

La syntaxe des conditions élémentaires est la suivante :

```
> age < 18 # age est strictement plus petit que 18
> age <= 18 # age est plus petit ou égal à 18
> age == 18 # age est égal a 18
ATTENTION : age = 18 ne donne pas une condition ! Il faut impérativement doubler le
signe "="
> age != 18 # age est différent de 18
```

Les conditions peuvent également être des objets complexes, des objets composés d'une combinaison de conditions.

```
> a <- TRUE
> b <- FALSE
> a && b # a ET b. Est vrai si a est vrai ET que b est vrai.
> a || b # a || b. Est vrai si a est vrai OU que b est vrai.
>!a # Non a. Est vrai quand a est faux
> (!a || (d<=35)) &&!(c >=d) # Est vrai quand... Heu... A faire en exercice !
```

3.3 Exercice

Ecrire une fonction qui prend une note (entre 0 et 20) et affiche la mention correspondante.

4 Boucles

Un deuxième pilier de la programmation est le concept de boucle. Une boucle est une instruction qui va exécuter une action plusieurs fois. La syntaxe générale est

`Pour <i prenant les successivement les valeurs x1, x2 puis x3>, faire <action>`

Sous **R**, la syntaxe est `for (i in 1 :10){<action>}`
A notez, on peut remplacer le vecteur 1 :10 par n'importe quel vecteur, comme par exemple c(1,3,5 :7)

```
> somme <- 0
> produit <- 1
> for (i in c(1,3,5 :7)){
>   cat("I = ",i,"\n")
>   somme <- somme + i
>   produit <- produit * i
>   cat("Fin de la boucle\n")
> }
```

On peut naturellement imbriquer une boucle dans une boucle :

```
> for (i in 0 :9){
>   for (j in 0 :i){
>    cat("X",j)
>   }
>   cat("X\n")
>}
```

4.1 Exercices

Écrire une fonction qui affiche un compte a rebours : on lui donne un nombre (par exemple 5), elle affiche `5... 4... 3... 2... 1... Partez !`

Écrire une fonction qui calcule C_n^p SANS utiliser la fonction `factorial()`

On dispose d'une distribution :

```
> dist <- c(6,5,5,4,4,4,4,rep(3,8),rep(2,16),rep(1,32),rep(0,64))
```

Tracez la distribution d'échantillonnage des moyennes des groupes de taille 10.
Indice : On tire aléatoirement un groupe de 10 personnes dans cette distribution. On calcule sa moyenne. On le stocke quelque part, on recommence 10000 fois et on trace la courbe obtenue.

5 Programmation propre

5.1 Nom des variables

5.2 Indentation

Qu'est ce qui ne marche pas dans le code suivant ?

```
> for (i in 0 :9){for (j in 0 :i){cat("X",j)cat("X\n")}
```

6 Débuggage