Niveau : M1 IS
Cours : Introduction au logiciel R
Enseignant : Aude Illig

Université Versailles Saint-Quentin
Année 2014/2015 - 1er semestre

TP 2 : Programmation en R

**** Manipulation des objets* `list` *et* `data.frame` *****

Table des matières

1 Boucles et vectorisation

R est un logiciel qui permet de programmer simplement l'éxécution successive de plusieurs commandes. Comme les autres langages, il dispose de boucles et de structures de contrôle (for, $while$, $repeat$, $if \ldots else$, ...) analogues à celles du langage C.

1.1 Structures de contrôle

La structure générale du branchement conditionnel `if` est la suivante :

```
if (condition) {
instruction(s)
} else
{
instruction(s)
}
```

Remarque. La partie `else` est facultative.

Exemple 1. Le programme suivant (à exécuter ligne à ligne) permet de calculer le prix TTC d'un produit à partir de son prix HT et du taux de TVA applicable :

```
> print("Saisie du prix (HT) :")
> ht<-scan()
> print("Saisie du type de TVA : 1- normal, autre- réduit)
> typetva<-scan()
> #Calcul du prix TTC
> if (typetva==1){
+ tauxtva=0.2
+ } else
+ {
+ tauxtva=0.055
+ }
> ttc <- ht * (1 + tauxtva)
> # Affichage explicite
> print(paste("Prix ttc = ",as.character(ttc)))
```

On peut simplifier la structure conditionnelle à l'aide de la commande `ifelse` pour renvoyer une valeur

```
> ifelse(typetva==1,0.2,0.055)
```

ou effectuer une affectation

```
> ifelse(typetva==1,ttc<-ht*1.2, ttc<-ht*1.055)
```

Remarque. La commande `ifelse` possède un autre avantage. La condition peut être vectorielle :

```
> typetva=c(1:2,1:3)
> ifelse(typetva==1,ttc<-ht*1.2, ttc<-ht*1.055)
```

1.2 Itérations

La boucle "pour" s'écrit ainsi :

```
for (indice in séquence) {
instruction(s)
}
```

Remarque. La boucle peut être interrompue avec la commande `break`.

Exemple 2. Formons à partir d'un vecteur `x` un autre vecteur `y` qui prend la valeur 1 pour les valeurs de `x` égales à 3 et la valeur 0 sinon :

```
> x=c(3,2,-5,3,4,3)
> y=numeric(length(x))
> séquence=1:length(x)
> for (i in séquence){
+ if (x[i]==3){
+ y[i]=1
+ } else
+ {
+ y[i]=0
+ }
+ }
> print(y)
```

Exemple 3. Ajoutons les vecteurs `x` et `y` pour former un nouveau vecteur `z` et créons un vecteur `t` constitué des carrés des composantes de `x` :

```
> z=numeric(length(x))
> t=numeric(length(x))
> for (i in 1:length(x)){
+ z[i]=x[i]+y[i]
+ t[i]=x[i]^2
+ }
> print(z)
> print(t)
```

La boucle "tant que" est structurée de la manière suivante :

```
while (condition) {
instruction(s)
}
```

Exemple 4. Augmentons de manière aléatoire la valeur d'un réel `a` jusqu'à dépasser la valeur arbitraire 10 :

```
> a=0.0
> bool=TRUE
> while (bool==TRUE){
+ a=a+runif(1)
+ if (a>10){
+ bool=FALSE
+ }
+ }
```

Remarque. Dans ces exemples, on constate que le prompt `>` de R est remplacé par `+` après l'ouverture d'une boucle par l'accolade `{` et ceci jusqu' à la fermeture de la boucle par l'accolade `}`.

1.3 Vectorisation

Dans la plupart des cas, les boucles peuvent (et doivent) être évitées au moyen d'une écriture vectorielle comme dans le cas de l'*Exemple 2.* :

```
> z=x+y
```

ou par une indexation logique comme dans le cas de l'*Exemple 1.* :

```
> y[x==3]=1
> y[x!=3]=0
```

Il est également possible de faire appel à la commande `apply(X,margin,fun)` où `X` est une matrice, `margin` indique si l'opération doit être appliquée sur les lignes (`margin=1`), sur les colonnes (`margin=2`) ou sur les deux (`margin=c(1,2)`) et `fun` est une fonction (c.f. paragraphe suivant) ou une opération :

```
> X=cbind(x,y)
> Z=apply(X,1,sum)
```

Les fonctions `sapply(X,fun)` pour les vecteurs et `lapply(X,fun)` pour les listes ou permettent d'appliquer une fonction `fun` ou une opération à chaque composante de `X` :

```
> square=function(x){x^2}
> T=sapply(x,square)     # sapply renvoie un vecteur si possible
> TT=lapply(x,square)    # lapply renvoie une liste plutôt qu'un vecteur
> dfX=data.frame(X)
> sapply(X,sum)          # sapply fonctionne sur chaque colonne du data.frame
```

La fonction `tapply(X,gpr,fun)` permet quand à elle d'appliquer une fonction `fun` aux sous-groupes d'un vecteur `X` définis à l'aide d'une variable `gpr` de type `factor` :

```
> xgpr=factor(y)
> tapply(x,xgpr,sum)
```

2 Procédures et fonctions

2.1 Ecriture de fonctions R

Les manipulations en R se font principalement au moyen de fonctions dont les arguments sont indiqués entre parenthèses. Tout utilisateur de R peut aussi créer ses propres fonctions qui viendront se rajouter le temps de la session à celles déjà enregistrées dans le logiciel. La fonction suivante permet de calculer l'IMC d'une personne de poids `p` (en kilos) et de taille `t` (en mètres) :

```
> IMC=function(p,t){
+ # Fonction calculant l'IMC d'une personne
+ # de poids p et de taille t
+ x=p/t^2
+ return(x)}
```

Remarque. Une fonction sans `return()` est appelée procédure, elle renvoie la dernière valeur calculée.

On appelle ensuite la fonction `IMC` :

```
> IMC(60,1.7)       # Choix des paramètres par position
> IMC(t=1.7,p=60)   # ou par nom
```

2.2 Utilisation de la classe `list`

A priori une fonction ne renvoie qu'un seul résultat. On peut cependant contourner le problème en utilisant la classe `list` de R. Ainsi, une fonction peut renvoyer une liste d'objets de classes différentes.

```
> ls()              # IMC est visible comme objet en mémoire
> rm(IMC)           # On efface la fonction IMC pour l'améliorer
> IMC=function(p,t,resume=TRUE){
> x=p/t^2
> if (resume==TRUE){
> return(list(poids=p,taille=t,indice=x))
> } else
> {
> return(x)
> }
> }
```

et on appelle la nouvelle fonction IMC :

```
> info=IMC(60,1.7)  # Affichage complet par défaut sous forme de liste
> info$indice       # Affichage de la variable indice de la liste info
> names(info)       # Renvoie les noms des objets de la liste
> IMC(60,17,FALSE)  # Affichage court par changement du paramètre par défaut
```

2.3 Emploi de la commande `replicate`

La commande `replicate` permet de lancer `n` fois une opération ou une fonction et de collecter les résultats dans un vecteur. Le programme suivant permet de lancer un dé équilibré à 6 faces et d'afficher les résultats.

```
> lance.de=function(m=6){
> point=1+trunc(m*runif(1))
> return(point)}
> res=replicate(100,lance.de(6))
> print(table(res))
```

Remarque. La commande `replicate` permet d'éviter l'utilisation d'une boucle !

3 Exercices d'application

3.1 Suite de Fibonacci

La suite de Fibonacci est la suite $(x_n)_{n\in\mathbb{N}^*}$ vérifiant :

$$\left\{\begin{array}{lcl} x_1 & = & 1 \\ x_2 & = & 1 \\ x_n & = & x_{n-1}+x_{n-2} \end{array}\right.$$

1. En utilisant une boucle `for`, calculer les 12 premiers termes de la suite de Fibonacci.
2. Changez les deux premiers termes de la suite en 2 et 2 et calculer les 12 premiers termes de la nouvelle suite.
3. En utilisant une boucle `while`, listez tous les termes de la suite de Fibonacci inférieurs à 300.

4. Ecrire une fonction calculant les n pemiers termes de la suite de Fibonacci.
5. Calculer les 30 premiers termes de la suite de terme général $y_n = x_n/x_{n-1}$, $n = 2, \ldots \infty$.
6. La suite $(y_n)_{n\geq 2}$ est-elle convergente ? Calculer $\frac{1+\sqrt{5}}{2}$. Concluez.

3.2 Calcul d'intérêts

1. Ecrire une fonction calculant le montant des intérêts simples produit par un capital P placé pendant n années au taux d'intérêt annuel i.
2. Quel est le montant des intérêts simples produit par 100 euros placés pendant 3 ans à un taux d'intérêt de 4% ?
3. Ecrire une fonction calculant le montant des intérêts composés produit par un capital P placé pendant n années au taux d'intérêt annuel i.
4. Quel est le montant des intérêts composés produit par 100 euros placés pendant 3 ans à un taux d'intérêt de 4% ?

Remarque. On rappelle qu'un capital produit :
- des intérêts simples si les intérêts sont uniquement calculés sur ce capital.
- des intérêts composés si à la fin de chaque période, les intérêts générés sont ajoutés au capital pour produire de nouveaux intérêts. On dit aussi que les intérêts sont capitalisés.