Formation
Langage de programmation R - Initiation

Édouard Hirchaud
edouard.hirchaud@univ-nantes.fr
Floriane Simonet
floriane.simonet@univ-nantes.fr

26 Septembre 2013

Table des matières

1 L'environement RStudio **1**
1.1 Éléments de RStudio 1

2 Premiers Pas **1**
2.1 R est une calculette 1
2.2 Créer des objets et les utiliser 2
2.2.1 Vecteurs 2
2.2.2 Matrices et data frame 3

3 Importer exporter des données **4**

4 Graphiques **5**

5 Écrire et utiliser une fonction maison **6**

6 Programmation : les bases **6**
6.1 Les boucles 6
6.2 Les conditions 7

1 L'environement RStudio

1.1 Éléments de RStudio

Présentation des différentes fenêtres de RStudio et personnalisation.

2 Premiers Pas

2.1 R est une calculette

```
1 # Une opération simple
2 10 + 3
3 # Une Opération plus complexe
4 10 / (3+8) * 78
5 # les différents opérateurs sont :
6 # multiplication *
7 # addition +
8 # division /
9 # soustraction -
```

EXERCICE Calculer le temps passé dans votre dernière structure professionelle (université, labo, entreprise ...) puis diviser par la différence entre l'année en cours et votre année de naissance. Multiplier par 100 pour obtenir un pourcentage du temps passé dans cette structure.

```
# Stockage d'une valeur dans une variable
nombreX <- 50.8

# Accès à la valeur stockée dans la variable nombreX
# Utiliser l'auto complétion. Commencer par écrire "nomb" puis appuyer sur la  touche de tabulation.
nombreX

# stockage de la variable nombreY (le symbole <- est identique à =)
nombreY <- 7.4

# Utiliser l'auto complétion, si plusieurs solutions existent vous pouvez
# continuer de taper le nom de la variable pour que le choix soit plus
# restreint puis utiliser les flèches du clavier pour choisir la bonne
# variable. Une fois sélectionnée, appuyer sur entrée.
nombreX + nombreY

# Stockage du résultat de l'opération dans la variable sommeXY
#   Conseil : utiliser l'historique en appuyant sur la 'flèche haut'
#   on peut retrouver des commandes déjà écrites.
sommeXY <- nombreX + nombreY
```

EXERCICE Répéter l'exercice précédant mais en utilisant des variables.

2.2 Créer des objets et les utiliser

2.2.1 Vecteurs

```
# Création d'un vecteur numérique
monVecteur1 <- c(20.7676, -45, 78, 12)

# Remarque sur les symboles > et + au début de la ligne de commande
#
c(20, 45, 78,

# Séquence de nombres de 1 à 30
maSuite <- seq(from = 1, to = 30)

# Une séquence de nombres pairs
maSuitePaire <- seq(from = 2, to = 20, by = 2)


# Nous avons utilisé la fonction seq()
# Pour trouver à quoi servent les arguments, taper la commande
?seq

# La documentation de cette fonction s'affiche sur le panneau en bas à droite de RStudio
```

EXERCICE En utilisant la fonction *seq()*, créer un vecteur de nombres impairs allant de 7 à 77.

```
# La fonction rnorm() permet de créer des vecteurs avec des nombres
# aléatoires.
monVecteurAlea <- rnorm(45)
```

```
4
5  # Création d'un vecteur avec chaine de caractères.
6  monVecteurA <- c("Mut_1", "Mut_2", "Mut_3")
7
8  # La fonction rep() permet de répéter une chaine de caractères un certain
       nombre de fois.
9  rep("Mut", 4)
10
11 # La fonction paste() permet de coller des chaines de caractères en les sé
       parant par un caractère.
12 paste("Mut", 1, sep = "_")
```

EXERCICE En utilisant les fonctions *rep()*, *paste()* et *seq()* créer un vecteur identique à *monVecteurA*, mais allant de 1 à 20. Dans un premier temps utiliser des étapes intermédiares.

```
1  # Accès à l'élément d'indice 1
2  monVecteur1[1]
3
4  # Accéder aux éléments d'indice  3 , 4 puis 5.
5
6  # Accès aux éléments d'indice 1 à 3
7  monVecteur1[1:3]
8
9  #Accès aux éléments d'indice 1 3 4
10 monVecteur1[c(1, 3, 4)]
11
12 # Pour connaitre la longueur d'un vecteur on utilise la fonction length()
13 length(monVecteur1)
```

EXERCICE Trouver une manière d'accèder au dernier élément d'un vecteur.

```
1  # Opérations sur tous les éléments d'un vecteur
2  monVecteur1 + 5.5
3  log(monVecteur1)
4  log10(monVecteur1)
5  log2(monVecteur1)
6  log(momVecteur1, 2)
7  abs(monVecteur1)
8  round(monVecteur1)
9  round(monVecteur1, 2)
10
11 # Opérations qui travaillent sur tout un vecteur
12 sum(monVecteur1)
13 mean(monVecteur1)
14 median(monVecteur1)
15 sd(monVecteur1)
16 var(monVecteur1)
17 max(monVecteur1)
18 min(monVecteur1)
19 range(monVecteur1)
20
21 # Addition de deux vecteurs
22 monVecteur2 <- c(10, 100, 5, 2)
23 monVecteur1 + monVecteur2
```

2.2.2 Matrices et data frame

```r
# Matrice avec des nombres aléatoires
maMatrice <- matrix(rnorm(100), ncol = 10)

# S'informer sur la matrice
# voir les n premières lignes d'une matrice
head(maMatrice)
```

EXERCICE

- Combien de lignes la fonction head() affiche -t-elle par défaut ?
- Comment afficher plus ou moins de ligne ? Utiliser la documentation de la fonction head pour trouver.
- Comment voir la matrice à partir des dernières lignes ?

```r
#Dimension de la matrice : nombre de lignes et nombre de colonnes
dim(maMatrice)
```

EXERCICE

- Quel type d'objet renvoie la fonction dim()
- La fonction *ncol()* permet de nous renseigner sur le nombre de colonnes. Trouver une fonction similaire pour trouver le nombre de lignes
- Avec la fonction *dim()* afficher uniquement le nombre de lignes

```r
# Une matrice est un ensemble de vecteurs.
# Chaque colonne est un vecteur, ainsi que chaque ligne.
# Pour récupérer la 9ème colonne.
col9 <- maMatrice[, 9]
```

EXERCICE Recupérer les colonnes 1, 7 et 3 de la matrice *maMatrice*

```r
# La fonction colnames() permet d'obtenir le nom des colonnes d'une matrice
    ou d'un data.frame
colnames(maMatrice)

#Donner un nom aux colonnes
colnames(maMatrice) <- LETTERS[1:ncol(maMatrice)]
colnames(maMatrice)

#Type et classe d'objet#
typeof(maMatrice)
class(maMatrice)

#Transformer une matrice en data.frame
monDataFrame <- as.data.frame(maMatrice)

#Autre moyen d'accéder aux colonnes avec un data.frame
colonneC <- monDataFrame$C
```

3 Importer exporter des données

```r
#La fonction essentielle : read.delim()
#Sans aucun argument
mesEchantillons <- read.delim("samples.txt")

#Avec des arguments
mesEchantillons <- read.delim("samples.txt", row.names = 1, dec = ",",
    stringsAsFactor = FALSE, header = TRUE)
```

```r

#Observer le nom des colonnes

mesEchantillons <- read.delim("samples2.txt", row.names = 1, dec = ",",
    stringsAsFactor = FALSE, header = TRUE)

#Possible d'importer des données avec R studio, mais tous les arguments ne
    sont pas disponibles.

#Sauvegarde dans un fichier
#La fonction essentielle write.table
write.table(x = mesEchantillons, file = "mesEchantillons.txt", col.names =
    NA , row.names = TRUE, quote = FALSE, sep = "\t")
```

Exercice : retrouver les cinq erreurs de cette ligne de code.

```r
#Ligne de commande erronée.
matriceBug <- read.delin("sample.txt" ,  rown.names = 1  head = true)
```

4 Graphiques

```r
#la fonction plot()
x<-1:20
plot(x, x^2)

#Il existe plusieurs fonctions de base pour les graphiques.
# plot()
# hist()
# boxplot()
# Utilisation basique
boxplot(maMatrice)


#Arguments communs à toutes les fonctions
plot(x, x^2, xlim=c(0, 30), ylim=c(-100, 500), xlab="Variable x", ylab="
    Variable x au carré", main="Carré des valeurs de 1 à 20", cex.axis=1.5,
    cex.lab=1.5, cex.main=2, bty="l", pch=16)


# Sauvegarde dans un fichier image
# Dans l'onglet Plots : Export-'Save Plot As Image'
# File name :  boxPlot
# La même chose avec la commande :
png("boxPlot.png")
plot(x, x^2, xlim=c(0, 30), ylim=c(-100, 500), xlab="Variable x", ylab="
    Variable x au carré", main="Carré des valeurs de 1 à 20", cex.axis=1.5,
    cex.lab=1.5, cex.main=2, bty="l", pch=16, col = 2)

#Pour ajouter une légende
legend("topright", legend = "X", pch = 16, col = 2)

# Ferme la fenêtre graphique et enregistre le fichier.
dev.off()

# Il existe 71 paramètres pour affiner les graphiques
# Liste des 71 paramètres de la fonction par()
par()
```

5 Écrire et utiliser une fonction maison

Préparation Il est plus facile d'écrire des fonctions dans un ou plusieurs fichiers. Créer un nouveau fichier et le nommer *mesFonctions.R* La syntaxe pour écrire une fonction est la suivante :

```
maFonction <- function(argument1, argument2 = valeurParDefaut){

  #On utilise le nom des arguments comme variable pour faire des calculs et
     appeler des fonctions.
  resultatTemporaire <- argument1  + argument2

  #On continue différents traitements avec d'autres variables crées dans la
     fonction
  resultatTraiter <- uneAutreFonction(resultatTemporaire)

  #On retourne le resultat de notre fonction avec la fonction return()
  return(resultatTraiter)


}
```

Ennoncé : Créer la fonction *ingredientsPateAPizza* qui prend comme argument le nombre de pizza (par défaut 1). Le retour est un vecteur avec les quantités pour X pizzas. :

```
ingredientsPateAPizza(nbPizza = 1)
  Farine  Eau Levure
[1] 500 250 20
```

Les ingrédients pour une pizza
- farine : 500
- eau : 250
- levure : 20

Tester sa fonction : La fonction est écrite dans le fichier fonction.R, mais elle est encore inconnue dans le *workspace*. Pour la mettre en mémoire il faut executer la fonction *source()*.

```
source("fonction.R")
```

Si il n'y pas d'erreurs de syntaxe, la fonction sera reconnue par R. On le remarque sur la fenêtre Workspace.
On peut maintenant utiliser notre fonction de la même façon que n'importe quelle autre.

EXERCICE : Créer la fonction *repartIngr* qui prend comme arguments
- *vecIngred* Un vecteur de nombre dont chaque indice à le nom d'un ingrédient.
- *imageName* Une chaine de caractère qui correspond au nom du fichier image.
- *recipiesName* Une chaine de caractère qui correspond au nom de la recette.

Cette fonction doit créer un graphique avec la fonction *pie* et l'enregistrer sous le même nom que *imageName*

6 Programmation : les bases

6.1 Les boucles

Le mot *for* permet de répéter une instruction un certain nombre de fois.

```
x <- 1:20

for(i in 1:length(x)){

x + 100

}
```

EXERCICE : Dans un nouveau fichier de votre choix, écrire une boucle *for* exécutant 10 fois la fonction *ingredientsPateAPizza*

6.2 Les conditions

Les instructions *if* et *else* permettent d'executer des parties de codes sous certaines conditions.

```
if ( x > 0){
  print("x est positif")

}else{
  print("x est négatif")
}


if ( x > 0){
  print("x est positif")

}else{
  if(x != 0){
    print("x est négatif")
  }else{
    print("x est nul")
  }
}
```

Exercice : Dans la fonction *ingrédientsPateAPizza* ajouter une instruction *if else* qui vérifie que l'argument nbPizza est bien positif et non nul. Auquel cas, un message du style "Veillez saisir un nombre de pizza positif" devra s'afficher.