TP01 : Découverte de R - Exercices

Justine Guégan - j.guegan-ihu@icm-institute.org
Guillaume Meurice - guillaume.meurice@gustaveroussy.fr

27 octobre 2016

Manipulation des vecteurs, facteurs et matrices

Exercice 1 : Premier vecteur

Soit `a=c("lannister", "targaryen", "baratheon", "starck", "greyjoy")`

- Essayer de faire a[1:3]. Qu'obtenez-vous ?
- Essayer de supprimer les lannisters ?
- Essayer de supprimer les lannisters et les greyjoy ?
- Trier par ordre alphabetique. *indice : `sort` ?*

```
a=c("lannister", "targaryen", "baratheon", "starck", "greyjoy")
a[1:3]
a[-1]
a[-c(1,5)]
sort(a)
```

Exercice 2 : Combiner des vecteurs

1. Créez un vecteur `a` contenant tous les entiers de 1 à 100.
2. Créez un vecteur `b` contenant tous les entiers pairs de 2 à 100.
3. Uniquement en utilisant `a` et `b`, créez un vecteur `c` contenant les entiers **impairs** entre 1 et 100.

Indice: `seq`.

```
a = c(1:100)
b = seq(from = 2, to = 100, by = 2)
c = a[-b]
```

Exercice 3 : Facteurs

1. Définissez un facteur `fac = factor(c("a","b","b","b","a","b","a","a"))`.
2. Calculez le nombre de `"a"` et de `"b"` dans `fac` en utilisant les fonctions `which` et `length` et des opérateurs binaires (`==`).
3. Que permet de faire la fonction `table` ? Appliquez la à `fac`.
4. Quels sonts les modalités de ce facteurs ? *indice : `levels`*

```
fac = factor(c("a","b","b","b","a","b","a","a"))
length(which(fac == "a"))
length(which(fac == "b"))
table(fac)
levels(fac)
```

Exercice 4 : Matrices

1. Exécuter la commande `a = rep(0:1, 50)`. Qu'a-t-on fait ?
2. Utilisez `a` pour construire une matrice `A` à 10 lignes et 10 colonnes.
3. Utilisez la fonction `t` sur cette matrice pour créer une matrice `B`. Que s'est-il passé ?
4. Que se passe-t-il après l'opération `M = A+B` ? Quelle est la dimension de la matrice `M` ? *indices : `ncol`, `nrow`, `dim`*
5. Les commandes `A[1:5, ]` et `B[, 1:5]` permettent de récuperer respectivement les 5 premières lignes de `A` et les 5 dernières colonnes de `B`. Inspirez-vous de ces commandes pour récupérez les lignes de 1 de `A` et les colonnes de 0 de `B`.
6. Extrayez les `2` de la matrice `M`.

```
a = rep(0:1, 50)
A = matrix(a, ncol = 10, nrow = 10)
B = t(A)
M = A+B

line1 = A[seq(2,10,2),]
col0  = B[,seq(1,10,2)]

M2    = which(M==2)
```

Listes et tableaux de données

Exercice 5 : `liste` et `data.frame`

1. Créez une liste `x` contenant une variable aléatoire gaussienne de taille 10 appelée `a` et un vecteur contenant uniquement des 1 de taille 10 également. On peut accéder aux deux éléments de cette liste avec les commandes `x[[i]]` ou `x$nom_de_la_variable`. *Indice : `rnorm`.*
2. Créez un objet `y` qui est la transformation de cette liste en `data.frame`. On peut maintenant parcourir les éléments de chaque objet comme pour une matrice avec la commande `y[i,j]` ! *indice = `as.data.frame`*
3. Créez deux objets `z1` et `z2` contenant respectivement les 3 premières et les 3 dernières lignes de `y`. Quelle est la classe de ces deux objets ?
4. Rajoutez à la liste `x` un vecteur `alphabet` contenant les lettres de l'alphabet. *indice : `letters`*
5. Essayez de transformer de nouveau `x` en `data.frame`. Que se passe-t-il ?

```
x = list(
  a = rnorm(10),
  v = rep(1, 10)
)
x[["a"]]
x$a
y = as.data.frame(x)
y
z1 = y[c(1:3) ,]
z2 = y[c(8:10),]
class(z1)
class(z2)

x$alphabet = c(letters)
as.data.frame(x)
```

Fonctions

Exercice 6 : Les boucles `for`

1. Lisez l'aide sur la procédure permettant de réaliser des boucles indicées `for` (`help("for")`). *Remarque :* demander de l'aide sur cette procédure avec la syntaxe `?for` ne fonctionnera pas ! Pourquoi ?
2. Pour exemple, exécutez la boucle suivante `for (i in 1:10) print(i)`.
3. A l'aide d'une boucle, calculez la somme des entiers pairs compris entre 1 et 100.

```
for (i in 1:10) print(i)

somme = 0
for (i in seq(2,100,2)) {
  somme = somme + i
}
somme
```

Exercice 7 : Création de fonction

1. Exécutez les commandes `data(iris)` puis `str(iris)`. Nous venons de charger en mémoire l'un des nombreux jeux de données distribués avec R ! Profitez de l'aide sur ce jeu de données pour en apprendre un peu plus sur les fleurs (`?iris`) ! Tous les jeux de données disponibles avec l'installation de base de R sont accessibles en tapant `data()`.

2. Créez la fonction `moyenneET` suivante et décryptez la :

```
moyenneET = function(i) c(moy = mean(iris[,i]), et = sd(iris[,i]) )
```

3. Afficher l'aide de la fonction `apply`. En utilisant cette fonction, calculez la moyenne et l'écart type des colonnes numériques du dataset `iris`. Comparer le résultat avec celui obtenu par la foncion `moyenneET`.

Remarque : pour exécuter plusieurs commandes au sein d'une même fonction, il faut utiliser des accolades `{...}`. Par exemple

```
#1
data(iris)
str(iris)
moyenneET(2)

#2
moyenneET <- function(i) {
  moy = mean(iris[,i])
  et  = sd(iris[,i])
  return( c(moy = moy, et = et) )
}


#3
?apply
moy = apply(iris[,c(1,2,3,4)], 2, mean)
et  = apply(iris[,c(1,2,3,4)], 2, sd)
```

```r
moyenneET(1)
moyenneET(2)
moyenneET(3)
moyenneET(4)
```

Exercice 8 : `for`, `if` et `else`

1. Comme dans l'exercice précédent, lisez l'aide de la procédure conditionnelle `if` : (`help("if")`).
2. Utilisez les structures `if` et `else` pour créer un programme qui prend en entrée un réel x et qui lui associe $y = x^2$ si x est positif et $y = x^3$ si x est négatif.
3. Utilisez les structures `for`, `if` et `else` pour créer un programme qui imprime à l'écran, pour chaque entier relatif i compris entre -10 et 10, i^3 si $i <= 0$, ou i^2 si $i > 0$.

```r
f1 = function(x){
  y = 0
  if (x > 0){
    y = x^2
  }
  else{
    y = x^3
  }
  return(y)
}

for (i in -10:10){
  print(paste(i, f1(i), sep="<=>"))
}
```

Exercice 9 : boucle while et for

1. Choisir un nombre mystère entre 1 et 100, et le stocker dans un objet que l'on nommera `nombre_mystere`. Ensuite, créer une boucle qui à chaque itération effectue un tirage aléatoire d'un entier compris entre 1 et 100. Tant que le nombre tiré est différent du nombre mystère, la boucle doit continuer. À la sortie de la boucle, une variable que l'on appellera `nb_tirages` contiendra le nombre de tirages réalisés pour obtenir le nombre mystère.
2. Utiliser le code de la question précédente pour réaliser la fonction `trouver_nombre`, qui, lorsqu'on lui donne un nombre compris entre 1 et 100, retourne le nombre de tirages aléatoires d'entiers compris entre 1 et 100 nécessaires avant de tirer le nombre mystère.
3. En utilisant une boucle for, faire appel 1000 fois à la fonction `trouver_nombre()` qui vient d'être créée. À chaque itération, stocker le résultat dans un élément d'un vecteur que l'on appellera `nb_essais_rep`. Enfin, afficher la moyenne du nombre de tirages nécessaires pour retrouver le nombre magique.

```r
# 1
nombre_mystere = 59

s = 0
nb_tirages = 0
while (s != nombre_mystere){
  s = sample(1:100, 1)
  nb_tirages  = nb_tirages + 1
}
```

```
nb_tirages

# 2
trouver_nombre = function(nombre_mystere){
  if (nombre_mystere >=1  & nombre_mystere <=100){
    s = 0
    nb_tirages = 0
    while (s != nombre_mystere){
      s = sample(1:100, 1)
      nb_tirages  = nb_tirages + 1
    }
    nb_tirages
  }
  else{
    warning("Votre nombre doit être compris entre 1 et 100")
  }
}

#3
nb_essais_rep = c()
for (i in seq(1000)){
  tmp = trouver_nombre(39)
  nb_essais_rep = c(nb_essais_rep, tmp)
}

mean(tmp)
```

Exercice 10 : Création d'une fonction, traitement de chaînes de caractères

Supposons que les adresses e-mails des étudiants de centralsupelec soient constituées de la manière suivante: le prénom et le nom de famille séparés par un point, le symbole arobase et enfin le nom de domaine. Supposons de plus que les étudiants ont un seul prénom, et aucune particule au nom de famille. La syntaxe des adresses e-mail est donc comme suit :

nom.prenom@etudiant.centralsupelec.fr.

1. Créer une fonction `parseMail`, qui à partir d'une adresse e-mail d'un étudiant, retourne un data.frame contenant trois variables : le prénom, le nom et l'adresse e-mail de cet étudiant.
2. Utiliser cete fonction pour créer un data.frame `emails.df` à partir du vecteur `emails`, contenant tous les prénoms, noms et adresses e-mail des étudiants

indice : `strsplit`

```
  emails = c( "john.snow@etudiant.centralsupelec.fr",
              "patti.smith@etudiant.centralsupelec.fr",
              "rick.grimes@etudiant.centralsupelec.fr",
              "mere.theresa@etudiant.centralsupelec.fr")
```

```r
#1
parseMail = function(email){
  nom_prenom = unlist(strsplit(email, "@"))[1]
  nom_prenom = unlist(strsplit(nom_prenom, "\\."))
  prenom = nom_prenom[1]
  nom = nom_prenom[2]
  data.frame(prenom = prenom, nom = nom, email = email)
}

#2
emails.df = c()
for (m in emails){
  df = parseMail(m)
  emails.df = rbind(emails.df,df)
}
```

Exercice 11 : Fonctions appliquées aux éléments d'une liste

Soit une liste nommée twittos, disponible à l'adresse suivante : XXXXXXXXXXXXX

Elle contient des informations fictives sur des utilisateurs de Twitter ; chaque élément de cette liste est une liste dont les éléments sont les suivants :

- screen_name: nom d'utilisateur
- nb_tweets: nombre de tweets
- nb_followers: nombre de followers
- nb_friends: nombre de followings
- created_at: date de création du compte
- location: ville renseignée

1. Importer le contenu du fichier dans la session R
2. Utiliser la fonction `lapply()` sur twittos pour récupérer une liste contenant uniquement les noms d'utilisateurs. Faire de même pour le nombre de followers, puis appliquer `unlist()` au résultat.
3. Créer une fonction qui, quand on lui fournit un élément de la liste twittos , c'est-à-dire les informations sous forme de liste d'un seul utilisateur, retourne ces informations sous forme de tableau de données. Nommer cette fonction `twittos_to_df`.
4. Appliquer la fonction `twittos_to_df()` au premier élément de la liste twittos, puis utiliser la fonction `lapply()` pour appliquer la fonction `twittos_to_df()` à tous les éléments de la liste. Stocker ce dernier résultat dans un objet appelé `res`
5. Quelle est la structure de l'objet `res` obtenu à la question précédente ?

```r
#1
load("twittos.rda")

#2
lapply(twittos, function(twit){twit$screen_name})

#3
twittos_to_df = function(twit){
```

```
  return(data.frame(twit))
}

#4
twittos_to_df(twittos[1])
res = lapply(twittos, twittos_to_df)
```

Lire et sauvegarder des données

Exercice 12 : Lire les données d'un fichier : fonction `read.table`

Il est possible de lire les données stockées dans des fichiers sous format `txt` grâce, entre autres, aux fonctions suivantes: `read.table()`, `read.csv()`, `read.csv2()` et `scan()`. Par ailleurs, la fonction `read.xls()` (resp. `write.xls()`) du package `gdata` fournit les outils pour lire (resp. écrire) des fichiers au format Excel.

(Récupérer les fichiers demandés sur le site XXXXXXXXXXXXXXX)

1. Importer dans une variable nommée `A` le jeu de données nommé `auto2004_original.txt`.
2. Importer dans une variable nommée `B` le jeu de données `auto2004_sans_nom.txt`.
3. Importer dans une variable nommée `C` le jeu de données `auto2004_virgule.txt`.
4. Importer dans une variable nommée `D` le jeu de données `auto2004_don_manquante.txt`. Combien de valeurs manquantes sont contenues dans le fichier ?
5. Importer dans une variable nommée `E` le jeu de données `auto2004_don_manquante(99999).txt`.
6. Importer dans une variable nommée `F` le jeu de données `bordeaux.xls`.
7. Quel est le mode des objets créés par la fonction `read.table()` ?

Indice : `help("read.table")`, `help("is.na")`

```
#1
A = read.table(file="auto2004_original.txt", sep="\t", header = TRUE)

#2
B = read.table(file="auto2004_sans_nom.txt", sep="\t", header = FALSE)

#3
C = read.table(file="auto2004_virgule.txt", sep="\t", header = TRUE, dec = ",")

#4
D = read.table(file="auto2004_don_manquante.txt", header = TRUE, sep="\t", na.strings = "")
nb = length(which(is.na(D) == TRUE))

#5
E = read.table(file="auto2004_don_manquante(99999).txt", header = TRUE, sep="\t", na.strings = "99999")

#6
F = read.xls(xls ="bordeaux.xls", header = TRUE)

#7
class(E)
```

Exercice 13 : Enregistrer des données

Créer la matrice suivante :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

1. Ecrire la matrice `A` dans un fichier nommé `matrice.txt`. Que remarquez vous?
2. Ajouter des arguments à la commande précédente pour retirer des noms aux lignes et aux colonnes du fichier créé.
3. Sauver la matrice `A` au format `.Rdata` dans le fichier `matriceA.Rdata` grâce à la fonction `save`.
4. Sauver la matrice `A` au format `.Rds` dans le fichier `matriceA.Rds` grâce à la fonction `saveRDS`.
5. Que donne la commande `C = load("matriceA.Rdata")` ?
6. Que donne la commande `D = readRDS("matriceA.Rds")` ?
7. Sauver toutes les variables dans un fichier nommés “données.Rdata”

```r
A = matrix(seq(12), ncol = 4, byrow = TRUE)

#1
write.table(A, file = "matrice.txt")

#2
write.table(A, file = "matrice.txt", row.names = FALSE, col.names = FALSE)

#3
save(A, file="matriceA.Rdata")

#4
saveRDS(A, file = "matriceA.Rds")

#5
C = load("matriceA.Rdata")
### la matrice A est rechargée.
### la variable C vaut "A"

#6
D = readRDS("matriceA.rds")
### la matrice A est rechargée dans la variable D

#8
save(list = ls(), "données.Rdata")
```