FACULTÉ DES SCIENCES ECONOMIQUES

UNIVERSITÉ DE RENNES 1

M1 Statistique & Économétrie

Ewen Gallic [1]
http://egallic.fr

2015

Logiciel R et programmation

Exercices

Partie 3 : Boucles

Exercice 1 (Boucle while)

1. À l'aide de la fonction `while()`, créer une boucle qui permet de calculer la factorielle d'un nombre ;

```
i <- 10
res <- 1
while(i>1){
  res <- res * i
  i <- i-1
}
```

2. Réutiliser le code de la question précédente pour en faire une fonction qui, lorsqu'on lui donne un nombre, retourne sa factorielle. Comparer le résultat avec la fonction `factorial()`.

```
# Fonction factorielle
# Retourne la factorielle de x
# @x : (int)
factorielle <- function(x){
  res <- 1
  while(x>1){
    res <- res * x
    x <- x-1
  }
  res
}# Fin de factorielle()

factorielle(100)
```

1. ewen.gallic[at]gmail.com

Exercice 2 (Boucles 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 ;

```r
nombre_mystere <- 30

nb_tirages <- 0
while(sample(x = seq_len(100), size = 1) != nombre_mystere){
  nb_tirages <- nb_tirages + 1
}
nb_essais
```

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 ;

```r
# Retourne le nombre de tirages aleatoires necessaires pour deviner
# correctement le nombre mystere (x)
# @x : (int) nombre mystere compris entre 1 et 100
trouver_nombre <- function(x){
  nb_tirages <- 1
  while(sample(x = seq_len(100), size = 1) != x){
    nb_tirages <- nb_tirages + 1
  }
  nb_tirages
}
```

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
nb_essais_rep <- rep(NA, 1000)
```

```r
for(i in seq_len(1000)) nb_essais_rep[i] <- trouver_nombre(10)

mean(nb_essais_rep)
```

Exercice 3 (Boucles for)

1. Parcourir les entiers de 1 à 20 à l'aide d'une boucle `for` en affichant dans la console à chaque itération si le nombre courant est pair ;

```r
library(stringr)
for(i in 1:20){
  if(i %% 2 == 0) print(str_c("Le nombre ", i, " est pair"))
}
```

2. L'objet `month.name` est un vecteur contenant les noms des mois du calendrier, en anglais. Parcourir chacun des éléments de ce vecteur, et afficher dans la console pour chacun des mois si le nombre de caractères composant le nom du mois est pair ou impair.

```
for(i in month.name){
  if(str_length(i) %% 2 == 0 ){
    # Le nombre de lettres du mois est paire
    cat(str_c("Dans \"", i,
                "\", il y a un nombre paire de caracteres\n"))
  }else{
    # Le nombre de lettres du mois est impaire
    cat(str_c("Dans \"", i,
          "\", il y a un nombre impaire de caracteres\n"))
  }
}
```

Exercice 4 (Suite de Fibonacci)

Utiliser une boucle `for` pour reproduire la suite de Fibonacci jusqu'à son dixième terme (la séquence F_n est définie par la relation de récurrence suivante : $F_n = F_{n-1} + F_{n-2}$; les valeurs initiales sont : $F_0 = 0$ et $F_1 = 1$).

```
end <- 10
res <- rep(NA, end)
res[1] <- 0
res[2] <- 1
for(i in 3:end){
  res[i] <- res[i-1] + res[i-2]
}
res
```

Exercice 5 (Barre de progression)

Considérons le vecteur de chaînes de caractères suivant :

```
library(magrittr)
n <- 1000
ids <-
  str_c(sample(LETTERS, n, replace = TRUE),
              sample(letters, n, replace = TRUE)) %>%
  unique()
```

Considérons également la liste `res` suivante :

```
res <- vector("list", length(ids))
```

Parcourir les éléments du vecteur `ids` à l'aide d'une boucle `for`. À chaque itération, stocker dans l'élément de la liste `res` dont la position correspond à celle de l'identifiant courant dans `ids` les informations suivantes : (i) l'identifiant courant et (ii) la somme de 50000 tirages aléatoires selon une $\mathcal{N}(0, 1)$.
Afficher l'état d'avancement de la boucle à l'aide d'une barre de progression.

```
p_b <- txtProgressBar(min = 1, max = length(ids), style = 3)
for(i in seq_len(length(res))){
  res[[i]] <- cbind(id = ids[i], value = sum(rnorm(50000)))
  setTxtProgressBar(p_b, i)
}
```

Exercice 6 (Fonctions appliquées aux éléments d'une liste)
Soit une liste nommée `twittos`, disponible à l'adresse suivante : http://egallic.fr/Enseignement/R/Exercices/donnees/twittos.rda. 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` ;

```
load(url("http://egallic.fr/Enseignement/R/Exercices/donnees/twittos.rda"))
```

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 ;

```
screen_names <- lapply(twittos, function(x) x$screen_name)
nb_followers <- lapply(twittos, function(x) x$nb_followers)
unlist(nb_followers)
```

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` ;

```
# A partir d'une liste concernant un Twittos,
# retourne ses informations sous forme de data.frame
# @x : (list) liste contenant des informations a propos d'un Twittos
twittos_to_df <- function(x){
  data.frame(screen_name = x$screen_name,
             nb_tweets = x$nb_tweets,
             nb_followers = x$nb_followers,
             nb_friends = x$nb_friends,
             created_at = x$created_at,
             location = x$location)
}# Fin de 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` ;

```
twittos_to_df(twittos[[1]])
res <- lapply(twittos, twittos_to_df)
```

5. Quelle est la structure de l'objet `res` obtenu à la question précédente ?

Il s'agit d'une liste.

```
class(res)
```

6. Utiliser la fonction appropriée dans le *package* `dplyr` pour transformer `res` en tableau de données ;

```
library(dplyr)
bind_rows(res)
```

7. Importer le fichier disponible à cette adresse dans la session `R` : http://egallic.fr/Enseignement/R/Exercices/donnees/dates_tw.rda. Il s'agit d'une liste donc chaque élément contient une liste indiquant le nom d'un utilisateur et la date de chacun de ses tweets.

```
load(url("http://egallic.fr/Enseignement/R/Exercices/donnees/dates_tw.rda"))
```

8. Appliquer la fonction `lapply()` à la liste `dates_tw` qui vient d'être importée dans `R`, pour afficher l'heure moyenne des tweets pour chaque utilisateur, puis faire de même pour l'écart-type.

```
lapply(dates_tw, function(x) hour(x$tweet_dates) %>% mean())
lapply(dates_tw, function(x) hour(x$tweet_dates) %>% sd())
```

Exercice 7 (Fonctions appliquées aux éléments d'une matrice)

1. Créer une matrice de dimension 100×5, donc chaque vecteur colonne est composé de tirages issus d'une loi Normale centrée réduite ;

```
m <- replicate(rnorm(100), n = 5)
```

2. Utiliser la fonction `apply()` pour calculer la moyenne des valeurs de chaque colonne ;

```
apply(m, 2, mean)
```

3. Utiliser la fonction `apply()` pour calculer l'écart-type des valeurs de chaque colonne.

```
apply(m, 2, sd)
```