Travaux dirigés pour l'introduction au logiciel R

Christophe Ambroise & Marco Pascucci

11/10/2018

les fonctions `apply`, `map` et `reduce`

FONCTION `apply()`

Fonctions appliquées à une dimension d'un tableau

Exercice 1 :

1. Créer une matrice de dimension 100x5, dont chaque vecteur colonne est composé de tirages issus d'une loi Normale centrée réduite ;
2. Utiliser la fonction `apply()` pour calculer la moyenne des valeurs de chaque colonne ;
3. Utiliser la fonction `apply()` pour calculer l'écart-type des valeurs de chaque ligne;

conseil : La fonction apply()

- afficher la documentation de la fonction `apply()` avec la touche **F1**
- Quelle est la dimention attendue pour le resultat du points 2? et celui du point 3?

dimensions des resultats

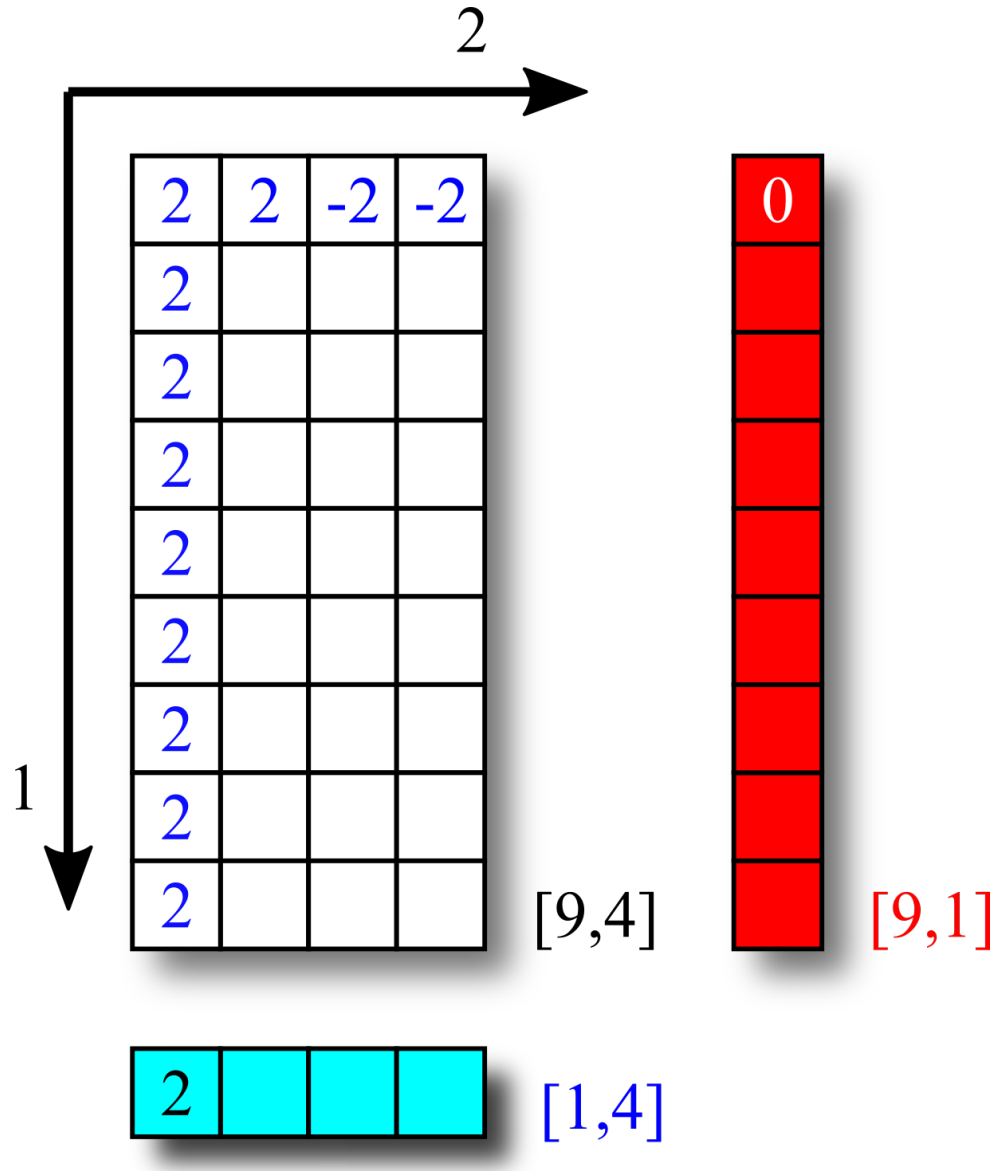


Figure 1: Moyenne par ligne ou colonne

Solution Ex.1

```r
# 1. Créer une matrice de dimension 100x5,
# dont 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 ;
meanM_cols <- apply(M, 2, mean)

# 3. Utiliser la fonction apply() pour calculer
# l'écart-type des valeurs de chaque ligne.
sdM_rows <- apply(M, 1, sd)
```

FONCTION `lapply()` (list apply)

Fonctions appliquées aux éléments d'une liste

RAPPEL :

A **vector** is a sequence of data elements of the same basic type.

```
my_vector <- c(1,3,2,5)
```

A **list** is a generic vector containing other objects.

```
my_list <- list(v=c(1,3,2), fruit="apple", number=0)
```

d'après la documentation de `lapply()` (**F1**)

"lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X."

`lapply()` dimensions

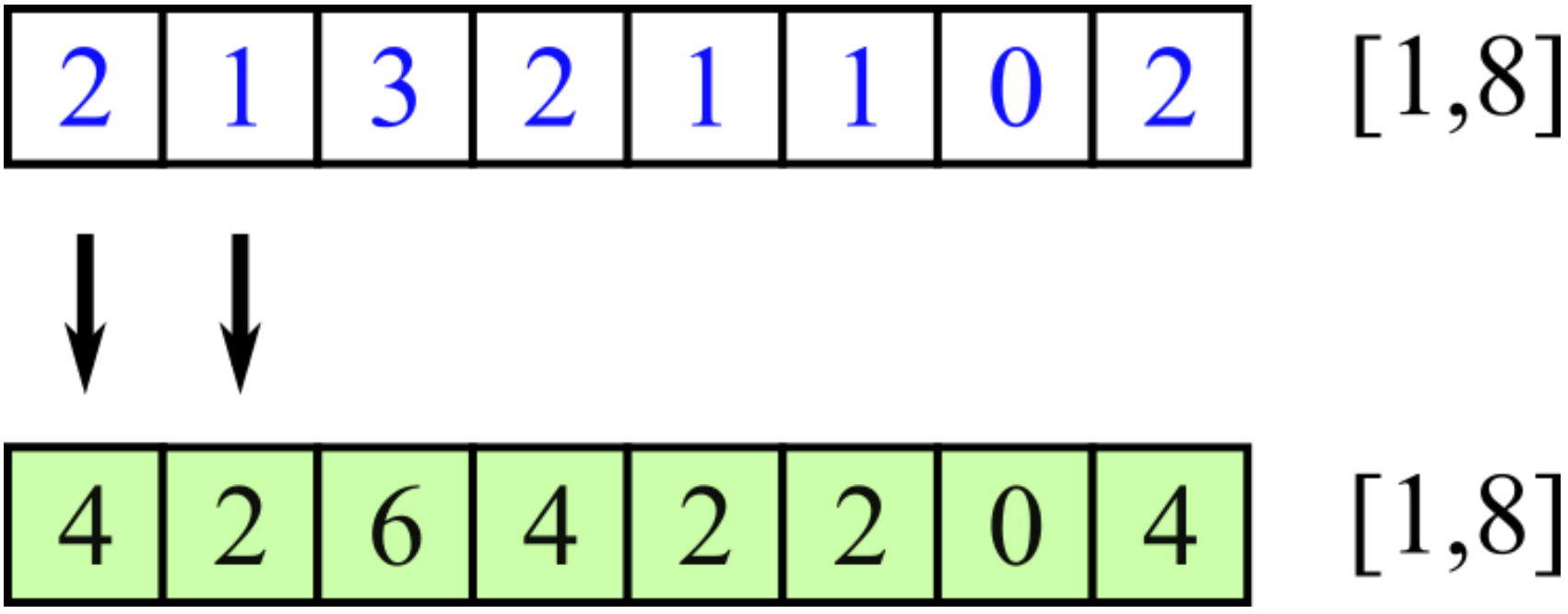


Figure 2: double element par element

Exemple `lapply()` “double”

```
v = list(2,1,3,2,1,1,0,2)
str(lapply(v, function(x) x*2))
```

```
## List of 8
##  $ : num 4
##  $ : num 2
##  $ : num 6
##  $ : num 4
##  $ : num 2
##  $ : num 2
##  $ : num 0
##  $ : num 4
```

Exercice 2

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. Charger les données
2. Décrire la classe et la structure de l’objet de twittos

Solution

```
# 1. Charger les données
load(url(url_of_twittos))
```

un objet twittos est créé (see Environment window)

```
# afficher le contenu de la liste comme texte
str(twittos)

# afficher la classe de l'objet twittos
class(twittos)
```

Exercice 2 (suite)

3. 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 ;
4. 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 ;
5. 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` ;
6. Quelle est la structure de l'objet res obtenu à la question précédente ?
7. Utiliser la fonction appropriée dans le package dplyr pour transformer res en tableau de données ;

Solution ex. 2.1-3

```r
# acceder a un element de la liste, eg. l'element 2
str(twittos[2])
# acceder a une proprieté nommée
twittos[[2]]$screen_name
```

```r
# selectionner une proprieté dans twittos
screen_names <- lapply(twittos, function(x) x$screen_name)
nb_followers <- lapply(twittos, function(x) x$nb_followers)
# transformer la liste en vecteur
nb_followers <- unlist(nb_followers)
```

Verifier la classe de `nb_followers` **avant** et **après** `unlist()`
Maintenant essayez d'utiliser `sapply()` (simplified lapply) à la place de apply.

Solution ex. 2.4

créer une fonction qui, quand on lui fournit un élément de laliste twittos, c'est-à-dire les informations sous forme de listed'un seul utilisateur, retourne ces informations sous forme detableau de données. Nommer cette fonction twittos_to_df

```r
# fonction qui genère un tableau (dataframe)
twittos_to_df <- function(x){
  # definition du dataframe
  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
  )
}
```

Solution ex. 2.5-6

```r
# appliquer la fonction à un élément
twittos_to_df(twittos[[1]])

# l'appliquer à toute la liste twittos
res <- lapply(twittos, twittos_to_df)

# afficher la classe de l'objet res et de ses elements
class(res)
class(nb_followers)
```

Solution ex 2.7

La fonction que nous allons utiliser est `bind_rows`. Mais elle n'est pas disponible par default dans R.

Cette fonction se trouve dans le *package* `dplyr` que nous devons avant tout importer.

```
# import dplyr package
library(dplyr)
res <- bind_rows(res)
```

Lisez la doc de la fonction `bind_rows` (**F1**) Qu'est-ce qu'elle fait? Observez maintenant la classe et structure de l'objet res...

Exercice 3 `lapply()`

1. 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.
2. 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.

Solution ex. 3

Il faut tout d'abord importer deux librairies:

```
library(tidyverse)
library(lubridate)
```

puis

```
# charger les données
load(url(url_of_dates_tw))
# calculer moyenne et SD avec lapply()
lapply(dates_tw, function(x) hour(x$tweet_dates) %>% mean()
lapply(dates_tw, function(x) hour(x$tweet_dates) %>% sd())
```

`Map()` or `mapply()`

fonctions pour listes ou vecteurs multiples

d'après la doc de MAP (**F1**)

Map applies a function to the corresponding elements of given vectors

Exercice 4

1. Charger le jeu de données `crabs` de `library(MASS)`
2. Visualiser les scatter plot en mettant en évidence les 4 groupes formés par le sexe et l'espèce des crabes.
3. Créer avec la fonction `Map` une variable qui décrive chacun des 4 groupes.
4. Présenter un résumé numérique de chacun des 4 groupes.

Solution ex. 4.1

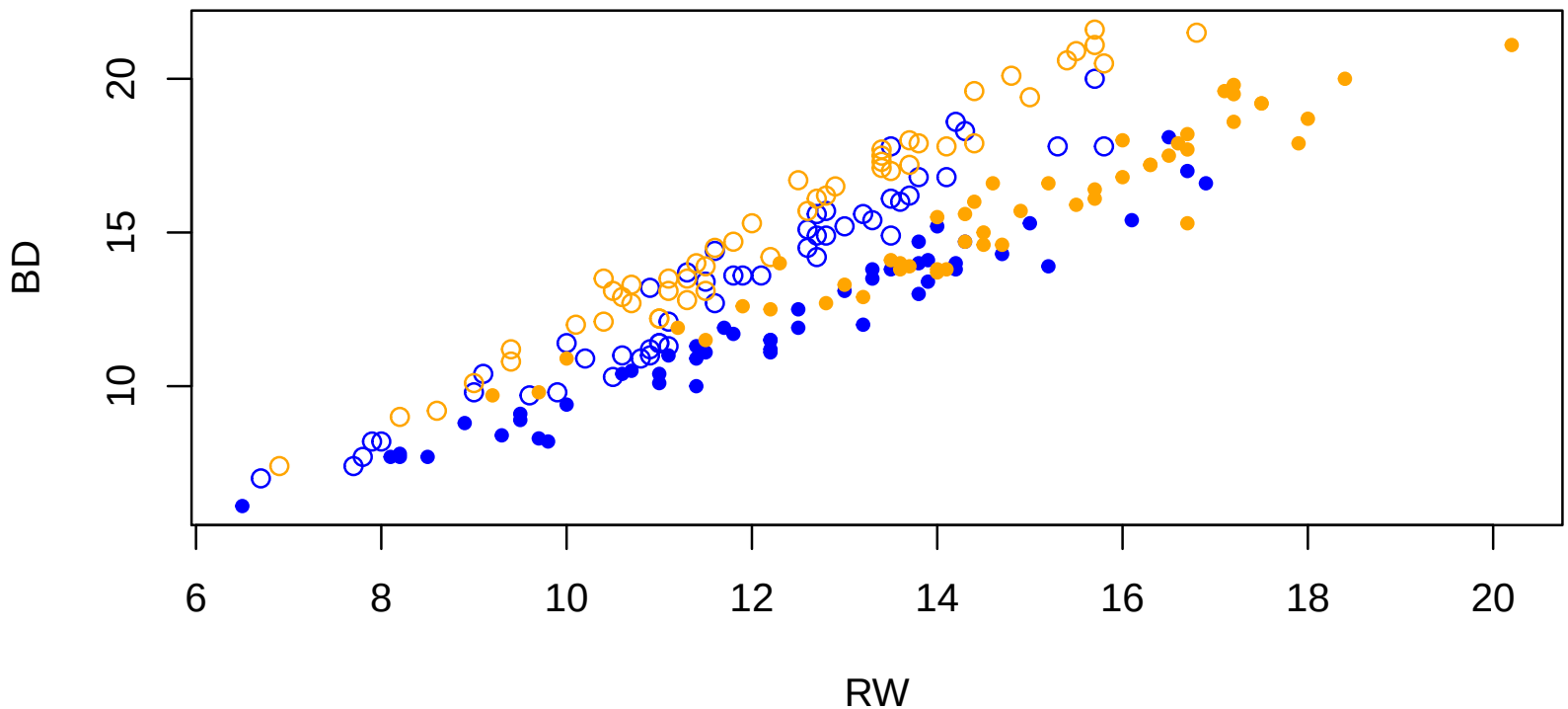
```
# importer le package MASS
library(MASS)
```

```
# charger les données crabs
data(crabs)
```

NOTE : les données ne sont pas encore chargées. Elle seront chargées seulement lors de leur prémière utilisation.

Solution ex. 4.2

```
# representer les données dans un scatter plot
plot(crabs[,c(5,8)],                          # data
     pch=c(20,21)[crabs$sex],                 # marker type
     col=c("blue","orange")[crabs$sp]         # color
     )
```



Solution ex. 4.3

```
label = Map(
  # fonction à mapper
  function(x,y) paste(x,y,sep="-"),
  # paramètres x et y
  crabs$sp,
  crabs$sex
  )
```

Quel est la classe de label? Et ses dimensions? Combien de labels differentes existent?

Solution ex. 4.4

4. Présenter un résumé numérique de chacun des 4 groupes.

```
by(
  data=crabs,          # data
  unlist(label),       # indices
  summary              # function to apply
  )
```

Un resumé est affiché.

Pour comprendre la solution de l'ex. 4.4

`summary()` affiche des information sur un objet comme un dataframe. Vous pouvez essayer d'afficher `summary(crabs)`.

`by()` execute summary sur crabs, après l'avoir regroupé selon les etiquettes definies dans `lable`. On a donc 4 application de summary, une pour chaque type d'etiquette dans `lable`.

La fonction `reduce()`

Appliquer une fonction a une liste en la consommant element par element, pour la reduire à une valeur

fonctionnement

Si on applique avec `reduce` une fonction f(x,y) sur une liste de 4 elements (x1,x2,x3,x4), le resultat sera:

```
f(x4,
  f(x3,
    f(x1,x2)
    )
  )
```

Exemple: somme

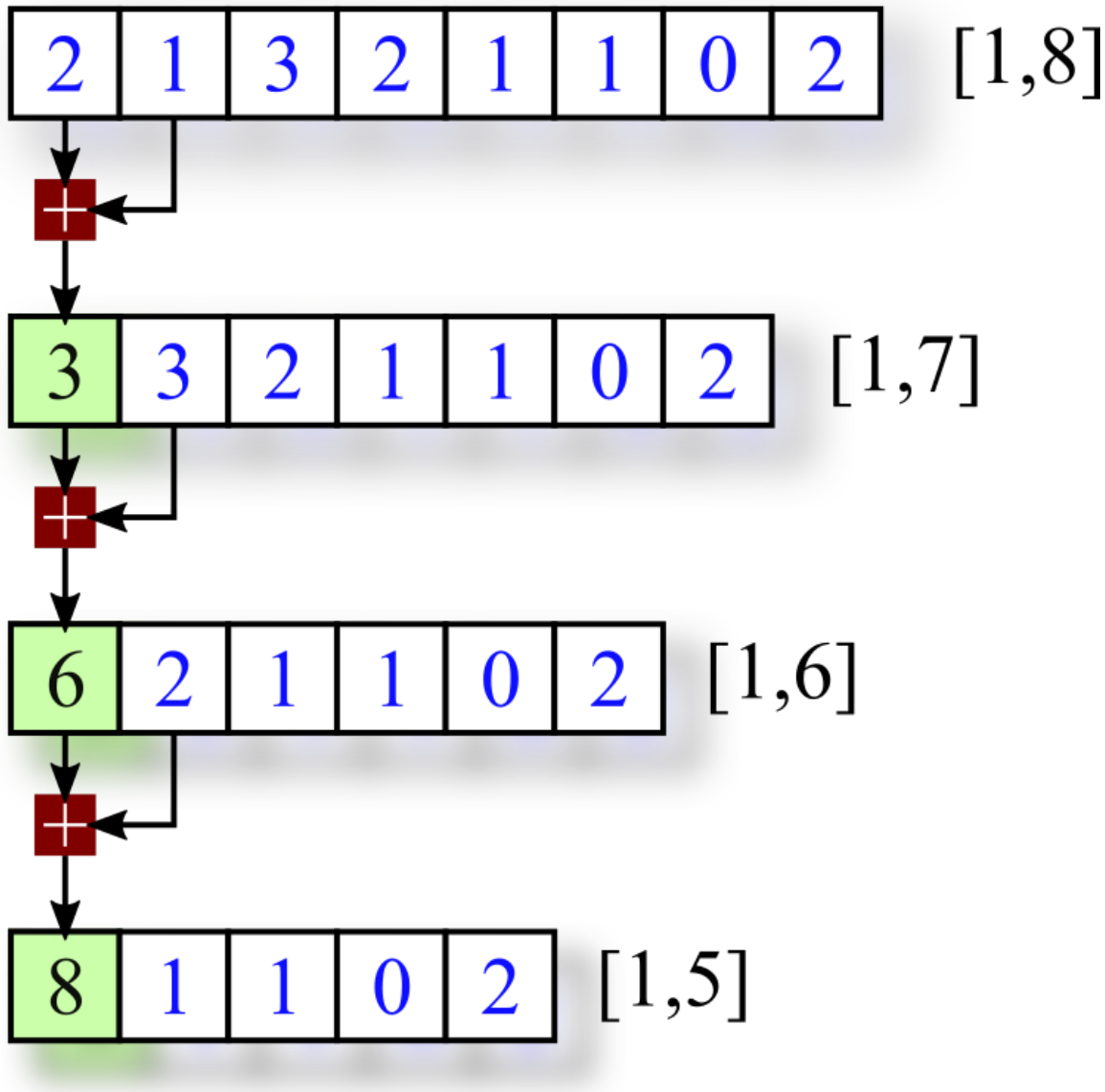


Figure 3: somme d'une liste d'entiers avec `reduce`

Exercice 5

En utilisant la fonction `reduce`

1. Calculer la somme des elements de la liste de la figure précedente.
2. Calculer le resultat de la multiplication des élément de la même liste. Quel est le resultat du point 2? Pourquoi?
3. Écrire une fonction qui determine si un vecteur est croissant ou pas, en utilisant seulement `Map()` et `reduce()`. Un vecteur est croiassant si chaque element est plus petit du suivant.

Solution ex. 5.1-2

```
# importer la librerie purrr qui contient reduce()
library(purrr)
```

```
# definir la liste
v = list(2,1,3,2,1,1,0,2)
# somme
reduce(v, function(x,y) x+y)
```

```
## [1] 12
```

```
# produit
reduce(v, function(x,y) x*y)
```

```
## [1] 0
```

Solution ex 5.3

```r
is_increasing <- function(v) {
  reduce(
    Map(
      function(x,y) x<y,  # la fonction de Map
      v[-length(v)],
      v[-1]
    ), function(x,y) x && y  # la fonction de reduce
  )
}

ok = c(1,2,3,4,5,6,7,8,9)
ko = c(1,2,3,1,5,6,7,8,9)

is_increasing(ok)  # TRUE
is_increasing(ko)  # FALSE
```