Les fonctions en R

Kofivi YENA || CoinDataConsulting

Table des matières

1 Pourquoi utiliser des fonctions ? 2

2 Création d'une fonction 2
2.1 Exemple 2 : une fonction qui calcule l'air d'un rectangle 2
2.2 Les arguments et valeurs par défaut 3

3 Les fonctions anonymes (lambda) 3
3.1 Exemple avec sapply : 3
3.2 Exemple avec apply 4
3.3 Exemple avec lapply(): 5
3.4 Exemple avec tapply 5
3.5 Exemple avec mapply 6

4 Comparaison de la famille apply 6

Dans ce tutoriel, nous allons apprendre à créer nos propres fonctions en R , un élément essentiel pour structurer et réutiliser du code efficacement.

Au programme :

- Pourquoi utiliser des fonctions ?
- Créer une fonction en R
- Les fonctions anonymes (lambda) : apply, lapply, tapply, mapply
- Comparaison entre les fonctions anonymes

1 Pourquoi utiliser des fonctions ?

Les fonctions permettent de :

Eviter la répétition de code

Rendre le code plus lisible et structuré

Faciliter la maintenance

Exemple : plutôt que de copier-coller une même opération plusieurs fois, on la met dans une fonction et on l'appelle à chaque fois qu'on en a besoin.

2 Création d'une fonction

nom_de_fonction <- function(argument1, argument2) { # Corps de la fonction resultat <- argument1 + argument2 return(resultat) # Valeur retournée } ## Exemple 1 : une fonction qui calcule la somme de deux nombres

```
addition <- function(a, b) {
  return(a + b)
}

# Appel de la fonction
addition(5, 3)  # Résultat : 8
```

```
[1] 8
```

2.1 Exemple 2 : une fonction qui calcule l'air d'un rectangle

Explication : La fonction prend deux arguments longueuret largeur. Elle retourne leur produit.

```
aire_rectangle <- function(longueur, largeur) {
  return(longueur * largeur)
}

# Utilisation
aire_rectangle(5, 3)  # Résultat : 15
```

```
[1] 15
```

2.2 Les arguments et valeurs par défaut

On peut attribuer des valeurs par défaut aux arguments :

```r
bonjour <- function(nom = "utilisateur") {
  print(paste("Bonjour", nom, "!"))
}

# Appels de la fonction
bonjour("Alice")  # Bonjour Alice !
```

```
[1] "Bonjour Alice !"
```

```r
bonjour()         # Bonjour utilisateur !
```

```
[1] "Bonjour utilisateur !"
```

3 Les fonctions anonymes (lambda)

Une fonction anonyme est une fonction sans nom , souvent utilisée dans les appels rapides.

3.1 Exemple avec sapply :

```r
nombres <- c(1, 2, 3, 4, 5)

# Fonction anonyme pour calculer le carré
carres <- sapply(nombres, function(x) x^2)
print(carres)  # Résultat : 1 4 9 16 25
```

```
[1]  1  4  9 16 25
```

Explication : function(x) x^2 est une fonction anonyme qui élève x au carré. sapply() applique cette fonction sur chaque élément du vecteur nombres.

3.2 Exemple avec apply

apply() est une fonction qui s'applique aux lignes ou colonnes d'une matrice.

syntaxe :

apply(matrice, MARGIN, fonction)

MARGIN = 1→ appliquer la fonction aux lignes

MARGIN = 2→ appliquer la fonction aux colonnes

Exemple pratique : calcul de la somme sur les lignes ou colonnes d'une matrice

```
mat <- matrix(1:9, nrow = 3, ncol = 3)
print(mat)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
# Somme des lignes
somme_lignes <- apply(mat, 1, sum)
print(somme_lignes)
```

```
[1] 12 15 18
```

```
# Somme des colonnes
somme_colonnes <- apply(mat, 2, sum)
print(somme_colonnes)
```

```
[1]  6 15 24
```

Explication :

apply(mat, 1, sum): additionne les valeurs de chaque ligne

apply(mat, 2, sum): additionne les valeurs de chaque colonne

3.3 Exemple avec lapply():

Appliquer une fonction à une liste (Retourne une liste)

Syntaxe :

lapply(liste, fonction)

Exemple : Calcul du carré des éléments d'une liste

```r
nombres <- list(1, 2, 3, 4, 5)

# Carré avec lapply
carres <- lapply(nombres, function(x) x^2)
print(carres)  # Liste : [[1]] 1, [[2]] 4, ...
```

```
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25
```

Explication : lapply() retourne toujours une liste

3.4 Exemple avec tapply

tapply(): Appliquer une fonction sur des sous-groupes d'un vecteur

syntaxe : tapply(vecteur, facteur, fonction)

```r
# Moyenne des notes par groupe d'étudiants

notes <- c(12, 15, 14, 9, 10, 18, 16)
groupes <- c("A", "A", "B", "B", "A", "B", "A")

moyenne_par_groupe <- tapply(notes, groupes, mean)
print(moyenne_par_groupe)
```

```
       A        B
13.25000 13.66667
```

Explication : tapply() applique mean() par groupe (A et B).

3.5 Exemple avec mapply

mapply(): Appliquer une fonction à plusieurs vecteurs simultanément

syntaxe : mapply(fonction, vecteur1, vecteur2, ...)

```r
# Additionner élément par élément deux vecteurs
x <- c(1, 2, 3, 4)
y <- c(10, 20, 30, 40)

somme_vecteurs <- mapply(sum, x, y)
print(somme_vecteurs)  # Résultat : 11 22 33 44
```

```
[1] 11 22 33 44
```

Explication : mapply() applique sum() sur chaque paire (x[i], y[i]).

4 Comparaison de la famille apply

```r
table_flextable
```

Fonction	Entrée	Sortie	Description
apply()	Matrice	Vecteur ou Liste	Applique une fonction aux lignes ou colonnes d'une matrice
lapply()	Liste	Liste	Applique une fonction à chaque élément d'une liste
sapply()	Liste/Vecteur	Vecteur ou Liste	Version simplifiée de lapply(), retourne un vecteur si possible
tapply()	Vecteur + Facteur	Vecteur	Applique une fonction sur des sous-groupes d'un vecteur
mapply()	Plusieurs vecteurs	Vecteur	Applique une fonction à plusieurs vecteurs simultanément

Exercice pratique

Utiliser apply()pour calculer la moyenne des lignes d'une matrice et tapply()pour calculer la somme d'un vecteur selon des groupes.