R : Introduction à R — Analyse R | Cours 3

Ményssa Cherifa-Luron

2023-09-15

Contents

Introduction **1**

Organisation . . . 3

Ressources . . . 3

Contrôle de Flux **3**

Les conditions : if/else - ifelse(), switch . . . 3

`if`, `else if`, et `else`. . . . 3

ifelse() . . . 3

Switch . . . 4

Les boucles . . . 4

Boucle for: s'exécute un nombre déterminé de fois. . . . 4

Boucle `for` . . . 4

Boucle `while` . . . 5

Fonctions . . . 6

1. Fonction `calculer_moyenne` . . . 6

2. Fonction `distance_euclidienne` . . . 7

Type de variables . . . 8

Variable locale . . . 8

Variable globale . . . 8

Exercices **8**

Introduction

Comme dans tous les langages de programamtion, **les structures de contrôle permettent de diriger le flux d'exécution du code**. Les trois principales structures de contrôle en R sont les **conditions, les boucles et les fonctions**.

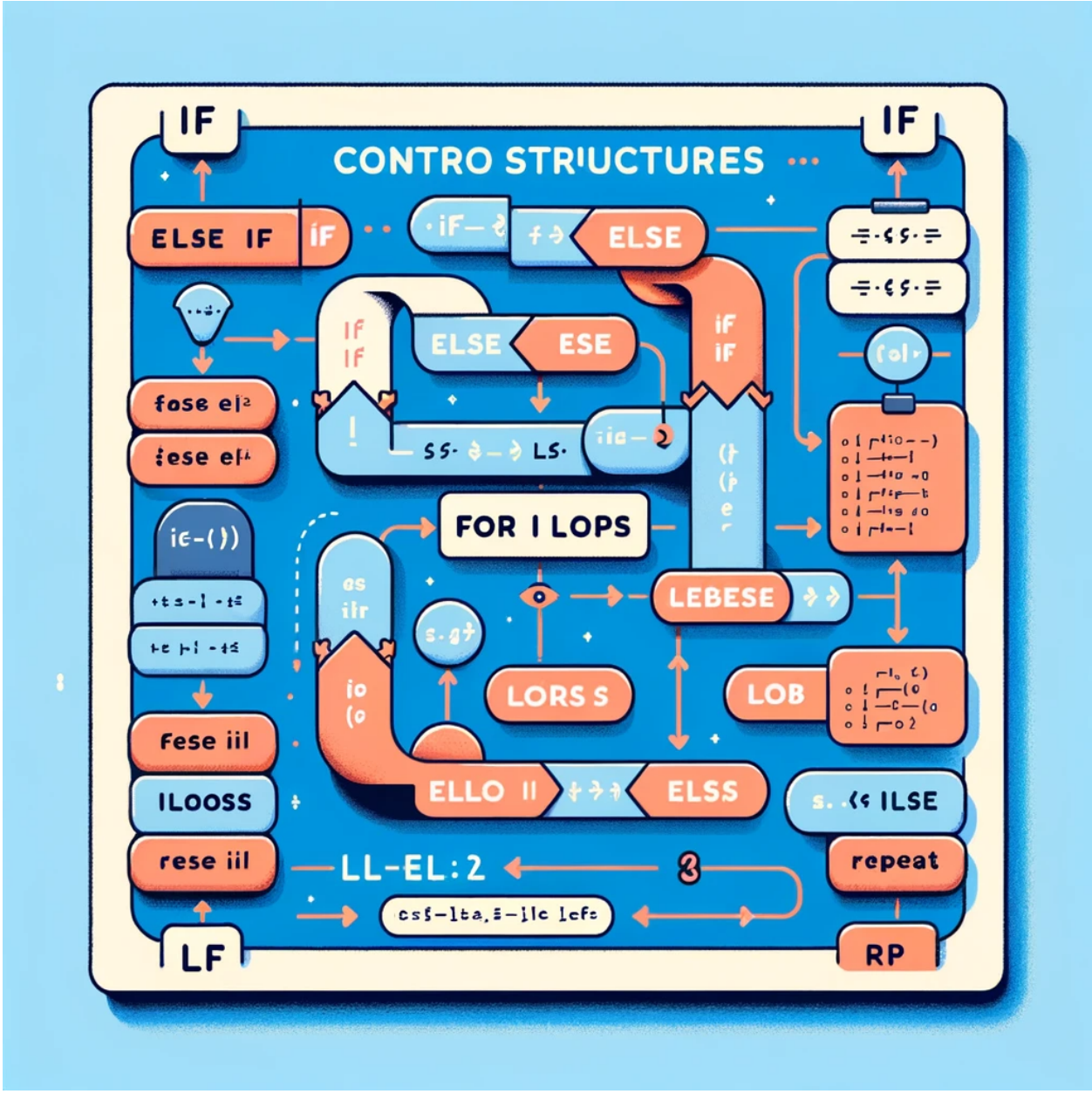

Figure 1: Illustration du langage R par DALL-E

Organisation

Contenu du Cours : - Démonstrations - Exercices

Ressources

- Rbloggers : Blog Populaire sur le langage R
- Datacamp : Plateforme d'apprentissage de la data science interactive
- Big Book of R : Edition qui rassemble des livres open-source sur le langage R
- Introduction accélérée au langage R pour la data science : L'essentiel et plus de tout ce que nous verrons

Contrôle de Flux

Les conditions : if/else - ifelse(), switch

Les instructions conditionnelles permettent d'exécuter différents blocs de code selon que certaines conditions sont remplies ou non.

`if`, `else if`, et `else`.

```
x <-  3

# Conditions (if, else)

if( x > 3){
  print("x est supérieur à 3")
}else if(x < 2){
  print("x est inférieur à 2")
}else{
  print("autre")
}
```

```
## [1] "autre"
```

ifelse()

La fonction **ifelse()** en R est une version vectorisée de la structure conditionnelle if-else. Elle est particulièrement utile pour effectuer des opérations conditionnelles sur des vecteurs.

```
# ifelse(condition, valeur_si_vrai, valeur_si_faux)
notes <- c(45, 75, 50, 60, 30)
mentions <- ifelse(notes >= 50, "Pass", "Fail")
print(mentions)
```

```
## [1] "Fail" "Pass" "Pass" "Pass" "Fail"
```

Switch

La structure de contrôle switch() est utilisée pour effectuer une sélection parmi plusieurs alternatives en fonction de la valeur d'une expression. Syntaxe : switch(expression, case1=value1, case2=value2, …)

```
fruit <- "pomme"

message <- switch(
  fruit,
  "pomme" = "C'est une pomme.",
  "banane" = "C'est une banane.",
  "orange" = "C'est une orange.",
  "raisin" = "C'est du raisin.",
  "Autre fruit."
)
```

Les boucles

Les boucles en R sont des structures de contrôle qui permettent de répéter un ensemble d'instructions plusieurs fois. Elles sont particulièrement utiles pour effectuer des opérations répétitives sans avoir besoin de réécrire le même code.

Boucle for: s'exécute un nombre déterminé de fois.

Les boucles en R sont des structures de contrôle qui permettent de répéter un ensemble d'instructions plusieurs fois. Elles sont particulièrement utiles pour effectuer des opérations répétitives sans avoir besoin de réécrire le même code.

L'objectif principal d'une boucle est la répétition. L'exécution du code se poursuit à l'intérieur de la boucle jusqu'à ce que la condition de sortie soit satisfaite.

Les variables déclarées à l'intérieur d'une boucle sont généralement locales à cette boucle (bien que cela puisse dépendre du langage).

Boucle `for`

- **Utilisation :** Une boucle `for` est utilisée pour exécuter un bloc de code un nombre déterminé de fois. Elle est particulièrement utile lorsque vous savez à l'avance combien de fois vous devez répéter les instructions.
- **Fonctionnement :**
  - `variable` est une variable temporaire qui prend successivement les valeurs présentes dans `sequence`.
  - `sequence` est typiquement un vecteur ou une liste.
  - À chaque itération, le bloc de code à l'intérieur de la boucle est exécuté avec la valeur courante de `variable`.
- **Exemple 1 :**

```r
x <- list(a = seq(1, 100, 4),
          b = seq(0, 1, 0.01),
          c = letters)

for (i in x){
  print(i)
}
```

```
##  [1]  1  5  9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
##   [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14
##  [16] 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29
##  [31] 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44
##  [46] 0.45 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59
##  [61] 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69 0.70 0.71 0.72 0.73 0.74
##  [76] 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89
##  [91] 0.90 0.91 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99 1.00
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

Cet exemple imprime les élèments dans la liste “x”.

- **Exemple 2 :**

```r
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Cet exemple imprime les nombres de 1 à 5.

Boucle `while`

- **Utilisation :** Une boucle `while` est utilisée pour répéter un bloc de code tant qu’une condition spécifiée reste vraie. Elle est utile quand le nombre de répétitions n’est pas connu à l’avance.
- **Fonctionnement :**
- La `condition` est évaluée avant chaque itération. Si elle est vraie (`TRUE`), le bloc de code est exécuté.
- La boucle continue jusqu’à ce que la `condition` devienne fausse (`FALSE`).
- **Exemple :**

```r
compteur <- 1

while (compteur <= 5) {
  print(compteur)
  compteur <- compteur + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Cet exemple imprime les nombres de 1 à 5, en incrémentant `compteur` à chaque itération.

Fonctions

Les fonctions sont des blocs de code réutilisables qui effectuent une tâche spécifique.

- **Définition de la fonction:**

```
# Fonction qui multiple deux valeurs a et b
multiplication <-  function(a, b){
  return(a * b)
}
```

```
- *multiplication* est le nom de la fonction.
- *function** est le mot-clé utilisé pour déclarer une fonction.
- *param1, param2, ...* sont les paramètres de la fonction, utilisés pour passer des données ou des val
```

Le corps de la fonction contient le code qui effectue la tâche. - *return(resultat)* est utilisé pour renvoyer une valeur du corps de la fonction à l'endroit où elle est appelée.

- **Appel de fonction:** Une fois définie, une fonction est appelée en utilisant son nom suivi de parenthèses contenant ses paramètres.

```
multiplication(a = 2, b = 4)
```

```
## [1] 8
```

- **Autres exemples:**

Le script R fourni comprend deux fonctions personnalisées, chacune ayant un objectif spécifique. Voici une explication détaillée de chaque fonction :

1. Fonction `calculer_moyenne`

Cette fonction calcule la moyenne d'un vecteur de nombres.

- **Structure de la Fonction :**

```r
calculer_moyenne <- function(vecteur) {
  # Vérification si le vecteur est vide
  if(length(vecteur) == 0) {
    warning("Le vecteur est vide!")
    return(NA)
  }

  # Calcul de la moyenne
  somme <- sum(vecteur)
  taille <- length(vecteur)
  moyenne <- somme / taille
  return(moyenne)
}
```

- **Fonctionnement :**
  - La fonction commence par vérifier si le vecteur passé en argument est vide. Si c'est le cas, elle affiche un avertissement et renvoie `NA`.
  - Si le vecteur n'est pas vide, la fonction calcule sa somme, détermine sa taille (nombre d'éléments), puis calcule la moyenne en divisant la somme par la taille.
  - Enfin, elle renvoie la valeur moyenne calculée.
- **Exemple d'Utilisation :**

```r
vecteur <- c(1, 2, 3, 4, 5)
print(calculer_moyenne(vecteur)) # Affiche la moyenne du vecteur
```

```
## [1] 3
```

```r
vecteur <- NULL
print(calculer_moyenne(vecteur)) # Affiche un avertissement et NA
```

```
## Warning in calculer_moyenne(vecteur): Le vecteur est vide!
```

```
## [1] NA
```

2. Fonction `distance_euclidienne`

Cette fonction calcule la distance euclidienne entre deux points dans un plan.

- **Structure de la Fonction :**

```r
distance_euclidienne <- function(point1, point2) {
  # Vérification des dimensions des points
  if(length(point1) != 2 || length(point2) != 2) {
    stop("Chaque point doit avoir exactement deux coordonnées!")
  }

  # Calcul de la distance
  delta_x <- point2[1] - point1[1]
  delta_y <- point2[2] - point1[2]
  distance <- sqrt(delta_x^2 + delta_y^2)
  return(distance)
}
```

- **Fonctionnement :**
  - La fonction vérifie d'abord si chaque point a exactement deux coordonnées (x, y). Si ce n'est pas le cas, elle arrête l'exécution avec un message d'erreur.
  - Elle calcule ensuite la différence entre les coordonnées x et y des deux points et utilise ces valeurs pour calculer la distance euclidienne (la racine carrée de la somme des carrés des différences).
- **Exemple d'Utilisation :**

```
pointA <- c(1, 2)
pointB <- c(4, 6)
print(distance_euclidienne(pointA, pointB)) # Affiche la distance euclidienne entre pointA et pointB
```

```
## [1] 5
```

Ces fonctions illustrent comment encapsuler des tâches spécifiques (calcul de moyenne, calcul de distance) dans des fonctions en R, permettant ainsi une réutilisation et une meilleure organisation du code.

L'exécution du code entre dans la fonction à son point d'appel, exécute les instructions à l'intérieur de la fonction, puis revient au point d'appel une fois la fonction terminée.

Les fonctions ont leur propre portée, ce qui signifie que les variables déclarées à l'intérieur d'une fonction ne sont généralement pas accessibles à l'extérieur de celle-ci.

Type de variables

Variable locale

C'est une variable déclarée à l'intérieur d'une fonction. Elle n'est accessible qu'à l'intérieur de cette fonction.

Variable globale

C'est une variable déclarée en dehors de toutes les fonctions. Elle est accessible partout dans le script.

```
variable_globale <- "Je suis globale !"

ma_fonction <- function() {
  variable_locale <- "Je suis locale !"
  print(variable_globale)
  print(variable_locale)
}

ma_fonction()
```

```
## [1] "Je suis globale !"
## [1] "Je suis locale !"
```

Exercices

Pour commencer à pratiquer, suivez ces étapes :

1. **Accédez au Dépôt GitHub :** Visitez l'URL fournie : https://github.com/universdesdonnees/Introduction-a-R pour accéder au dépôt GitHub contenant les matériaux du cours.

2. **Trouvez le Fichier des Exercices :** Dans le dépôt, localisez le fichier nommé **exercices3.txt**. Ce fichier contient les premiers exercices que vous devez pratiquer.

3. **Lisez et Essayez de Résoudre les Exercices :** Ouvrez le fichier **exercices1.txt** et lisez attentivement les exercices. Essayez de les résoudre par vous-même dans votre environnement R (comme RStudio). Il est important de pratiquer par vous-même avant de regarder les solutions pour mieux apprendre.

4. **Consultez la Correction :** Une fois que vous avez tenté de résoudre les exercices, ou si vous rencontrez des difficultés, consultez le fichier **correction_exercices3.R** pour voir les solutions. Analysez les solutions pour comprendre les méthodes et logiques utilisées.