



UNIVERSITÉ  
CÔTE D'AZUR

# Ensembles et dictionnaires

Algo & Prog avec R

---

A. Malapert, B. Martin, M. Pelleau, et J.-P. Roy

4 décembre 2025

Université Côte d'Azur, CNRS, I3S, France  
`firstname.lastname@univ-cotedazur.fr`

# Où en sommes-nous des types de données ?

## Types simples

```
> class(42)
[1] "numeric"
```

```
> class(
  as.integer(42))
[1] "integer"
```

```
> class(TRUE)
[1] "logical"
```

```
> class("42")
[1] "character"
```

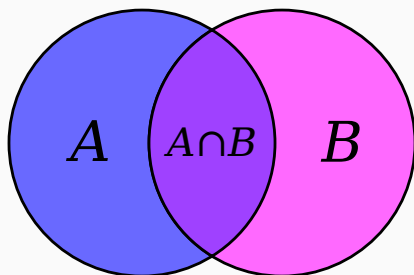
## Types composés : séquence d'objets numérotés (vector et list)

```
> class(c(1,2,3))
[1] "numeric"
```

```
> class(list('a',2,3))
[1] "list"
```

## Types composés : ensembles et dictionnaires.

- ▶ Disponible dans de nombreux langages, mais pas en R.
- ▶ Nous allons les émuler grâce à des vecteurs et des listes nommés.



**Figure 1:** Diagramme de Venn.

Origine : Georg Cantor (1845-1918), mathématicien allemand.

# Qu'est-ce qu'un ensemble ?

## Ensemble en R

Un ensemble en R est une séquence sans répétition et sans ordre (conceptuellement).

- ▶ En théorie, les éléments sont de types quelconques.
- ▶ En pratique, on représente un ensemble (type simple) par un vecteur.

## L'arithmétique des vecteurs ne suffit pas.

```
> 1:5 == 5:1 ## Vrai du point de vue ensembliste.  
[1] FALSE FALSE TRUE FALSE FALSE
```

## Égalité de deux ensembles

```
> setequal(1:5, 5:1)  
[1] TRUE
```

# Appartenance et cardinal d'un ensemble

## Appartenance d'un élément à un ensemble

```
> is.element(1, 1:5)
[1] TRUE
> is.element(0, 1:5)
[1] FALSE
```

## Cardinal d'un ensemble

Pas de fonction prédéfinie.

```
> Cardinal <- function(x) length(unique(x))
> Cardinal(c(1:5, 1:5))
[1] 5
```

# Opérations sur les ensembles

**Union**  $A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$

```
> union(1:5, 4:6)
[1] 1 2 3 4 5 6
```

**Intersection**  $A \cap B = \{x \mid x \in A \text{ et } x \in B\}$

```
> intersect(1:5, 4:6)
[1] 4 5
```

**Différence asymétrique**  $A - B = \{x \mid x \in A \text{ et } x \notin B\}$

```
> setdiff(1:5, 4:6)
[1] 1 2 3
```

**Inclusion**  $A \subseteq B \Leftrightarrow ((\forall x \in A) \Rightarrow (x \in B))$

```
> Inclusion <- function(x, y) setequal(x, intersect(x, y))
> Inclusions(5:6, 4:6)
[1] TRUE
```

# Qu'est-ce qu'un dictionnaire ?

## Dictionnaire en R

En théorie, un dictionnaire est une collection non numérotée de couples clé/valeur où la clé est un objet non mutable et la valeur est n'importe quelle valeur.

- ▶ Toutes les clés doivent être distinctes !
- ▶ R ne propose pas de classe pour les dictionnaires.

En pratique, nous allons **émuler un dictionnaire** dont les clés sont des chaînes de caractères avec un **vecteur nommé** ou une **liste nommée**.

## Vecteurs et listes nommés

- ▶ Écrire un code plus lisible.
- ▶ Utiliser les fonctions avancées de R (par ex. graphiques).

# Construction d'un dictionnaire

## Construction en extension

On fournit tous les couples dans un ordre quelconque.

```
> stock <- c(poires=51, pommes=243)
> print(stock)
poires  pommes
51      243
```

## Construction en deux temps

on fournit toutes les valeurs, puis tous les noms, dans le même ordre.

```
> stock <- c(51,243)
> names(stock) <- c("poires", "pommes")
```

Remarquez que nous affectons les noms aux résultats d'une fonction !



# Accès aux valeurs d'un dictionnaire

## Accès par clé

- ▶ On peut accéder à la valeur associée à une clé.
- ▶ La recherche est pratiquement instantanée ! De la clé vers la valeur !
- ▶ La recherche est unidirectionnelle : Français → Anglais.

```
> stock['pommes']  
250  
> getElement(stock, "poires")  
[1] 51
```

## Accès par rang

```
> stock[1]  
poires  
51
```

## Ne pas confondre rang et clé

```
> stock['1'] #ni clé, ni index  
<NA>  
NA
```

# Modification d'un dictionnaire

## Modifier la valeur associée à une clé

```
> stock["pommes"] <- 250
```

## Ajouter un nouveau couple clé/valeur

```
> stock["fraises"] <- 50  
> print(stock)  
poires  pommes  fraises  
51 250 50
```

## Attention, un dictionnaire est un vecteur !

```
> c(stock, stock) # Les clés ne sont plus uniques !  
poires  pommes  fraises  poires  pommes  fraises  
51      250      50      51      250      50
```

## Vérifier la présence d'une clé

```
> is.element("fraises", names(stock))  
[1] TRUE
```

# Itérer dans un dictionnaire

## Accéder aux valeurs et aux clés

```
> unname(stock) # vecteur des valeurs sans les clés  
[1] 51 250 50  
> names(stock) # vecteur des clés  
[1] "poires" "pommes" "fraises"
```

## Par défaut, on itère sur les valeurs

```
> for(v in stock) {print(v)}  
[1] 51  
[1] 250  
[1] 50
```

## Mais, on peut itérer sur les clés

```
> for(v in names(stock)) {print(v)}  
[1] "poires"  
[1] "pommes"  
[1] "fraises"
```

# Suppression dans un dictionnaire

## Suppression par clé

On ne peut pas directement supprimer la valeur associée à une clé.

```
> stock[-'pommes']  
Error in -"pommes" : invalid argument to unary operator
```

## Émuler la suppression par clé

On va devoir trouver le rang de la clé, puis la supprimer par rang. Pour cela, on définit une fonction `Remove(dict, key)` qui renvoie le dictionnaire après suppression du couple `key:val` du dictionnaire.

```
Remove <- function(dict, key) {  
  dict[ names(dict) != key]  
}
```

```
> Remove(stock, "pommes")  
poires fraises  
51 25
```

# De l'utilité des dictionnaire ...

On considère maintenant différentes variétés de chaque fruit.

On ne peut plus utiliser un vecteur. On doit utiliser une liste.

```
> stock <- list(  
  pommes = c(grany=25, golden=50, fuji=0),  
  poires = c(comice=10, williams=100),  
  fraises = c(marat=0, gariguette=100)  
)
```

## Accès par clé

```
> stock$fraises # accès par clé  
  marat gariguette  
    0         100  
  
> stock[["fraises"]] # accès par clé  
  marat gariguette  
    0         100  
  
> stock["fraises"] # extraction d'une tranche !  
$fraises  
  marat gariguette  
    0         100
```

# Pour regarnir ses étals

Trouver les variétés dont le nombre de fruit est en dessous d'un seuil fixé.

```
CommanderVarietes <- function(stock, seuil) {  
  commandes <- c()  
  for(fruit in stock) {  
    varietes <- names(subset(fruit, fruit < seuil))  
    commandes <- c(commandes, varietes)  
  }  
  return(commandes)  
}
```

```
> CommanderVarietes(stock, 50)  
[1] "grany"  "fuji"   "comice" "marat"  
> CommanderVarietes(stock, 10)  
[1] "fuji"   "marat"
```

Questions?

Retrouvez ce cours sur le site web

[`www.i3s.unice.fr/~malapert/R`](http://www.i3s.unice.fr/~malapert/R)