

Mise à niveau R 2/3

Eric Marcon

23 février 2024

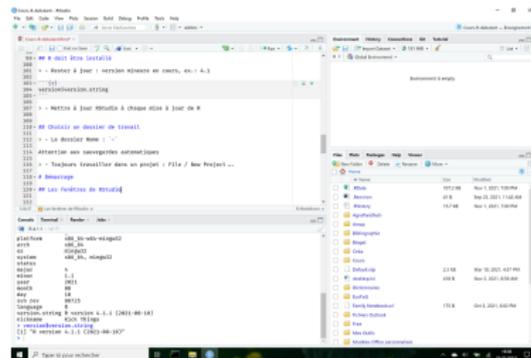
Données

Créer un script R. Ecrire en haut à gauche.

```
# Affectation
a <- 1
# ou encore
1 -> a
# mais éviter
a = 1
```

Exécuter (Ctrl+Entrée) : voir en bas à gauche.

Environnement en haut à droite.



Plutôt que des nombres :

```
x <- 1:5
```

```
2 * x
```

```
## [1] 2 4 6 8 10
```

```
sqrt(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```



Introduction à R et au tidyverse, [Prise en main](#)

Créer des vecteurs (1)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Des valeurs :

```
(x <- 1)
```

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

```
(x <- c("pommes", "poires"))
```

```
## [1] "pommes" "poires"
```

```
(x <- c(TRUE, FALSE))
```

```
## [1] TRUE FALSE
```

Créer des vecteurs (2)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Une séquence :

```
(x <- 1:5)
```

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

```
(x <- seq(from = 1, to = 5, by = 1))
```

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

Créer des vecteurs (3)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Une répétition :

```
(x <- rep(1, 5))
```

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

```
(x <- rep(1:2, each = 2))
```

```
## [1] 1 1 2 2
```



Utiliser systématiquement l'aide

```
?rep
```

Sélectionner des éléments (1)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Utiliser les crochets :

```
x <- (1:10) * 10  
x[3]
```

```
## [1] 30
```

```
x[-5]
```

```
## [1] 10 20 30 40 60 70 80 90 100
```

Utiliser des vecteurs pour sélectionner :

```
x[c(1, 3)]
```

```
## [1] 10 30
```

Tirer des nombres dans une séquence, trouver lesquels sont pairs.

```
x <- 1:100  
# Échantillonnage  
(y <- sample(x, 5))
```

```
## [1] 91 16 67 14 18
```

```
(y %% 2 == 0)
```

```
## [1] FALSE TRUE FALSE TRUE TRUE
```

Sélectionner des éléments (2)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Utiliser les crochets :

```
y[y %% 2 == 0]
```

```
## [1] 16 14 18
```

Les compter :

```
sum(y %% 2 == 0)
```

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

Les vecteurs contiennent des données de même mode :

- numérique : 1:2, 1L (L pour un entier)
- imaginaire : $(1 + 1i) * (1 - 1i)$ égale 2
- logique : TRUE
- caractère : "Bonjour"
- vide : NULL

Les matrices ont deux dimensions et contiennent des données de même mode

```
(A <- matrix(1:9, nrow = 3))
```

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

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

```
A[1, 2]
```

```
## [1] 4
```

```
A[, 3]
```

```
## [1] 7 8 9
```

Extension des matrices à plus de deux dimensions

```
A <- array(1:18, dim = c(3, 3, 2))
```

```
A
```

```
## , , 1
```

```
##
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    4    7
```

```
## [2,]    2    5    8
```

```
## [3,]    3    6    9
```

```
##
```

```
## , , 2
```

```
##
```

```
##      [,1] [,2] [,3]
```

```
## [1,]   10   13   16
```

```
## [2,]   11   14   17
```

```
## [3,]   12   15   18
```

Comme dans une matrice :

```
A[, , 2]
```

```
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

Eléments disparates :

```
(L <- list(noms=c("X", "Y"), tailles = c(100, 120)))
```

```
## $noms
## [1] "X" "Y"
##
## $tailles
## [1] 100 120
```

Sélectionner dans une liste

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Double crochet ou nom :

```
L[[2]]
```

```
## [1] 100 120
```

```
L$nom
```

```
## [1] "X" "Y"
```

Tableau dont chaque colonne est de mode unique :

```
(df <- data.frame(nom = c("X", "Y"), taille = c(100, 120)))
```

```
##   nom  taille
## 1   X     100
## 2   Y     120
```

Sélectionner dans un dataframe (1)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Sélectionner comme dans une matrice...

```
df[2, ]
```

```
##   nom taille  
## 2   Y     120
```

... ou comme dans une liste

```
df$taille
```

```
## [1] 100 120
```

Sélectionner dans un dataframe (2)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Sélection de lignes en fonction de valeurs

```
df[df$taille == 100, ]
```

```
##   nom  taille  
## 1   X     100
```

Fonctions

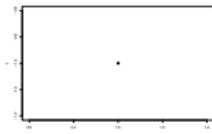
R est un langage fonctionnel.

```
y <- cos(pi)
```

Une fonction produit une valeur à partir d'arguments.

Une fonction peut avoir des effets de bord :

```
x <- plot(y)
```



```
x
```

```
## NULL
```

plot n'est utilisé que pour ses effets de bord.

Appeler une fonction en nommant tous ses arguments...

```
runif(n = 3, min = 0, max = 1)
```

```
## [1] 0.10293624 0.02248227 0.91641919
```

... ou en les passant dans l'ordre :

```
runif(3, 0, 1)
```

```
## [1] 0.9096517 0.1370688 0.3397247
```



Bonne pratique : nommer tous les arguments à partir du deuxième:

```
runif(3, min = 0, max = 1)
```

```
## [1] 0.1005619 0.4669571 0.4286474
```

Voir l'aide de la fonction : `?runif`

`min` et `max` ont des valeurs par défaut : 0 et 1.

```
runif(3)
```

```
## [1] 0.53216272 0.09179518 0.02105684
```

Syntaxe:

```
puissance <- function(x, r = 1) {  
  return(x^r)  
}  
puissance(1:3, r = 2)
```

```
## [1] 1 4 9
```

Penser vecteur. `r` est recyclé.

```
puissance(1:3, r = 3:1)
```

```
## [1] 1 4 3
```

Fonctions opérateurs (*infix functions*)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

Les opérateurs de R sont en fait des fonctions:

```
identical(2 + 2, `+`(2, 2))
```

```
## [1] TRUE
```

Les opérateurs définis par l'utilisateur sont obligatoirement entre % :

```
`%+%` <- function(a, b) paste(a, b)  
"Nouvelle" +% "chaîne"
```

```
## [1] "Nouvelle chaîne"
```

Référence

Les ... sont des arguments libres à passer à une autre fonction :

```
f <- function(x) x
g <- function(y, ...) f(...)
g("Rien", x = "Argument x passé à f par g")
```

```
## [1] "Argument x passé à f par g"
```

Mais il faut que tout argument soit reconnu par une fonction :

```
tryCatch(g("Rien", z = 2), error= function(e) print(e))
```

```
## <simpleError in f(...): unused argument (z = 2)>
```

Structures de contrôle

```
est_paire <- function(x) {  
  if (x %% 2 == 0) {  
    return(TRUE)  
  } else {  
    return(FALSE)  
  }  
}  
est_paire(3)
```

```
## [1] FALSE
```



Fonction non vectorielle. Utiliser plutôt :

```
((1:3) %% 2) == 0
```

```
## [1] FALSE TRUE FALSE
```

```
for (i in 1:3) {  
  print(sqrt(i))  
}
```

```
## [1] 1  
## [1] 1.414214  
## [1] 1.732051
```



Seulement si la fonction utilisée n'est pas vectorielle.

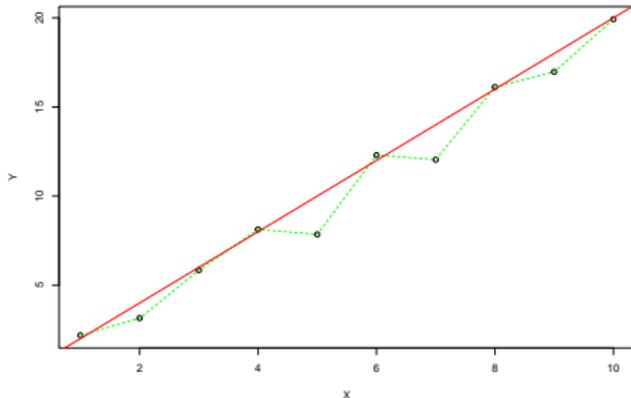
```
sqrt(1:3)
```

```
## [1] 1.000000 1.414214 1.732051
```

Graphiques de base

Graphiques simples :

```
X <- 1:10  
Y <- 2 * X + rnorm(length(X))  
plot(x = X, y = Y)  
lines(x = X, y = Y, col = "green", lty = 2)  
abline(a = 0, b = 2, col = "red")
```



Classes (1)

Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base

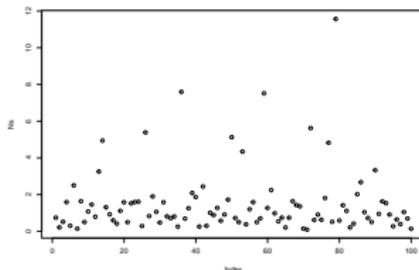
Les objets appartiennent à des classes.

```
Ns <- rlnorm(100)  
class(Ns)
```

```
## [1] "numeric"
```

plot est une méthode, déclinée par classe.

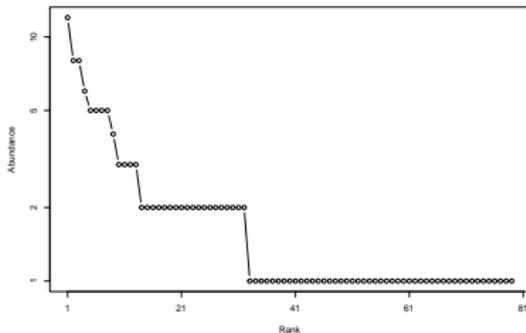
```
plot(Ns) # plot.numeric()
```



```
library("entropart")  
Ns <- as.AbdVector(Ns)  
class(Ns)
```

```
## [1] "AbdVector"                "SpeciesDistribution"  
## [3] "integer"
```

```
plot(Ns) # plot.SpeciesDistribution
```



Mise à niveau
R 2/3

Eric Marcon

Données

Fonctions

Structures de
contrôle

Graphiques de
base