7.1. Les vecteurs

Générez les vecteurs suivants, chacun de 2 façons différentes :

```
[1] 10.0 12.5 15.0
```

```
c(10.0,12.5,15.0)
seq(from= 10, to=15, by=2.5)
seq(from= 10, to=15, length.out=3)
```

```
[1] 15 16 17 18 19 20 15 16 17 18 19 20
```

```
c(15:20,15:20)
rep(15:20, 2)
```

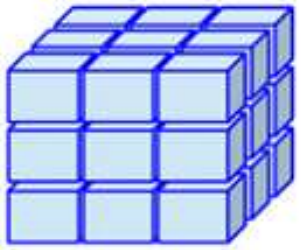
```
[1]  4  5  6  7 10  9  8
```

```
c(4:7,10:8)
c(seq(from= 4, to=7),seq(from= 10, to=8))
```

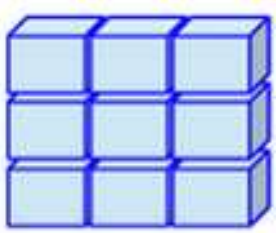
```
[1] "Rouge"  "Bleu"   "Rouge"  "Bleu"   "Rouge"  "Bleu"   "Vert"   "Vert"
    "Orange" "Orange" "Orange" "Orange"
```

```
c("Rouge", "Bleu", "Rouge", "Bleu",  "Rouge", "Bleu",
  "Vert", "Vert", "Orange", "Orange", "Orange", "Orange")

c(rep (c ("Rouge", "Bleu"), 3), rep("Vert",2),rep("Orange",4))
```

7.2. Les tableaux et matrices



Les **tableaux** (array) et les **matrices** (matrix) sont des vecteurs encapsulés de façon à ce qu'on puisse les manipuler en plusieurs dimensions (attribut **dim**). Tous leurs éléments sont de même type (numeric, character, logical).



Zoom sur les matrices; cas particulier de tableaux…

- Elles sont bidimensionnelles ; lignes x colonnes.
- Sont créées à partir d'1 ou plusieurs vecteurs dont les valeurs sont prises 1 par 1 pour remplir les cases de la matrice (intersection ligne x colonne).
- La dimension (nb lignes x nb colonnes) est donné par la fonction dim().
- Toutes les lignes et/ou colonnes ont le même nombre d'éléments.
- Disposent d'opérateurs spécifiques (inversion, produit matriciel, etc.).

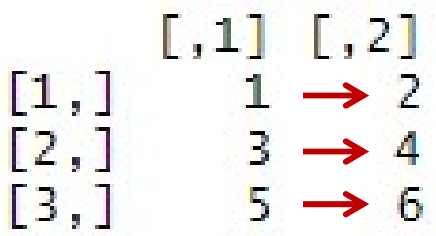
7.2. Les tableaux et matrices

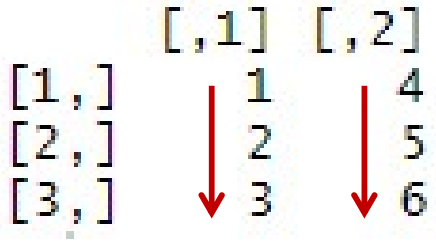
Différentes méthodes pour créer une matrice :

A partir de la fonction **matrix()** et d'**un vecteur** (x, crée précédemment):

Il faut spécifier le nombre de colonnes ncol et/ou de lignes nrow, ainsi que le sens de remplissage de la matrice (par ligne ou par colonne).

```
x           # vecteur à compiler; renvoie [1]  1  2  3  4  5  6

mat1 <- matrix (x, ncol = 2, nrow = 3, byrow = T)
mat1
class(mat1) ; dim(mat1)       # renvoie  [1]  "matrix " ; [1]   3  2

mat2 <- matrix (x, ncol = 2, byrow = F)
mat2
class(mat2) ; dim(mat2)       # renvoie  [1]  "matrix " ; [1]   3  2
```





7.2. Les tableaux et matrices

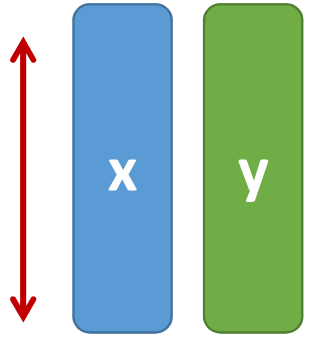
Différentes méthodes pour créer une matrice :

En combinant **plusieurs vecteurs** par colonne (**cbind()**) ou par ligne (**rbind()**) :

```
# Créons un second vecteur y
y <- c (7:12)   # class = numeric, length = 6

# avec cbind()
mat3 <- cbind (x, y)
mat3
dim (mat3)      # renvoie [1]   6  2 (nb L x nb C)
```

```
     x  y
[1,] 1  7
[2,] 2  8
[3,] 3  9
[4,] 4 10
[5,] 5 11
[6,] 6 12
```

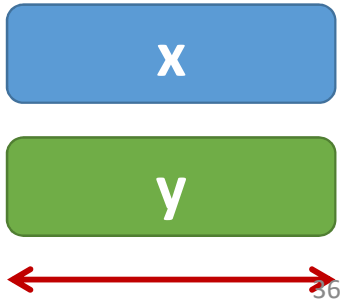
cbind() : association par **colonne**




```
 # avec rbind()
mat4 <- rbind (x, y)
mat4
dim (mat4)      # renvoie [1]   2  6
```

```
  [,1] [,2] [,3] [,4] [,5] [,6]
x    1    2    3    4    5    6
y    7    8    9   10   11   12
```

rbind() : association par **ligne**



7.2. Les tableaux et matrices

Changer le nom des lignes et colonnes d'une matrice avec rownames() et colnames().

```
### Exemple sur mat1
colnames(mat1) <- c("Var1","Var2")
rownames(mat1) <- c("Ech1","Ech2", "Ech3")
mat1
```

```
     Var1 Var2
Ech1    1    2
Ech2    3    4
Ech3    5    6
```

Intérêt : on peut s'appuyer sur les noms pour accéder au contenu de la matrice (indexation).

```
mat1[ ,"Var1"]


mat1["Ech3", ]
```

```
Ech1 Ech2 Ech3
   1    3    5

Var1 Var2
   5    6
```

7.2. Les tableaux et matrices

Quelques opérations matricielles utiles :

Les opérations +, -, * et / entre 2 matrices de même dimension sont des opérations terme à terme:

```
mat5 <- mat1 + mat2
mat5
```

```
      Var1 Var2
Ech1     2    6
Ech2     5    9
Ech3     8   12
```

La matrice transposée (lignes <-> colonnes) s'obtient avec t() :

```
mat6 <- t(mat5)
mat6
```

```
     Ech1 Ech2 Ech3
Var1    2    5    8
Var2    6    9   12
```

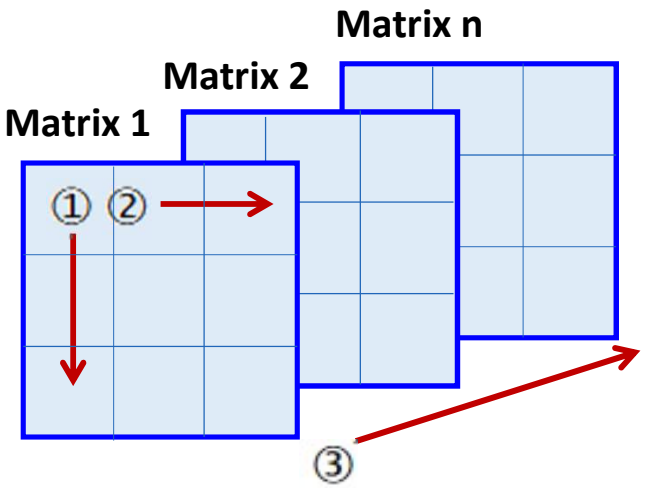
… et bien d'autres…

7.2. Les tableaux et matrices

(Pour information : Création d'un tableau (array))

Représentation schématique d'un tableau (array) :
Les chiffres donnent l'ordre de remplissage
(1: nb lignes, 2: nb colonnes, 3: nb n de matrices)

Matrice initiale




```
Array3D <- array ( 1:24, dim = c (3, 4, 2),

                   list ( c ("ROW1","ROW2","ROW3"),

                          c ("COL1","COL2","COL3", "COL4"),

                          c ("Matrix1","Matrix2") ) )
```

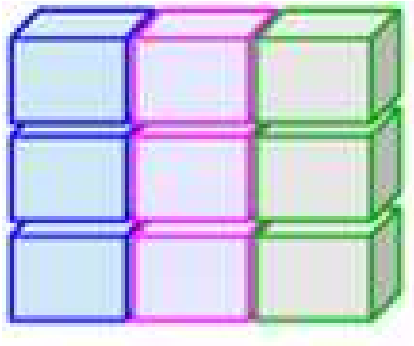
```
, , Matrix1

     COL1 COL2 COL3 COL4
ROW1    1    4    7   10
ROW2    2    5    8   11
ROW3    3    6    9   12

, , Matrix2

     COL1 COL2 COL3 COL4
ROW1   13   16   19   22
ROW2   14   17   20   23
ROW3   15   18   21   24
```

7.3. Les data frames

Structure très utilisée en statistique pour stocker des tableaux de données croisant des individus statistiques (observations, site, réplicat…) en lignes et des variables en colonnes.

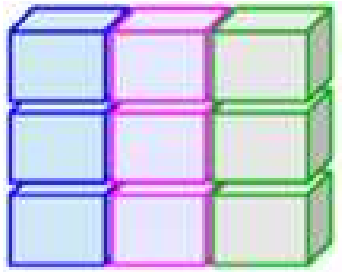
Peut être vu comme une matrice (**2D**) améliorée dans laquelle :

- Chaque colonne est un vecteur-variable dont tous les éléments sont du même type.
- L'ensemble des colonnes sont de même longueur, mais peuvent être de type différent (numeric, character, logical, factor).
- Chaque ligne (individu/observation) possède un identifiant unique.



7.3. Les data frames

Créer un data frame avec la fonction data.frame() :



```
site_ID <- c ("A1_01", "A1_02", "B1_01", "B1_02")
pH_sol <- c (5.6, 7.3, 4.1, 6.0)
nb_sp <- c (17, 23, 15, 7)
traitement <- c ("Fert", "Fert", "Non_Fert", "Non_Fert")

df <- data.frame ( pH_sol, nb_sp, traitement, row.names = site_ID)
df

class(df)     # renvoie  [1]  "data.frame"
dim(df)      # renvoie  [1]  4  3  (nb L x nb C)
```

	pH_sol	nb_sp	traitement
A1_01	5.6	17	Fert
A1_02	7.3	23	Fert
B1_01	4.1	15	Non_Fert
B1_02	6.0	7	Non_Fert

7.3. Les data frames

Bien souvent, vous importerez directement des fichiers pré-remplis au format .csv ou .txt …

read.csv() ou read.csv2()

read.table()

On reviendra sur cette structure de données et les fonctions associées :

header()

head()

str()

summary()

melt()

…

7.4. Les listes

(Pour information)

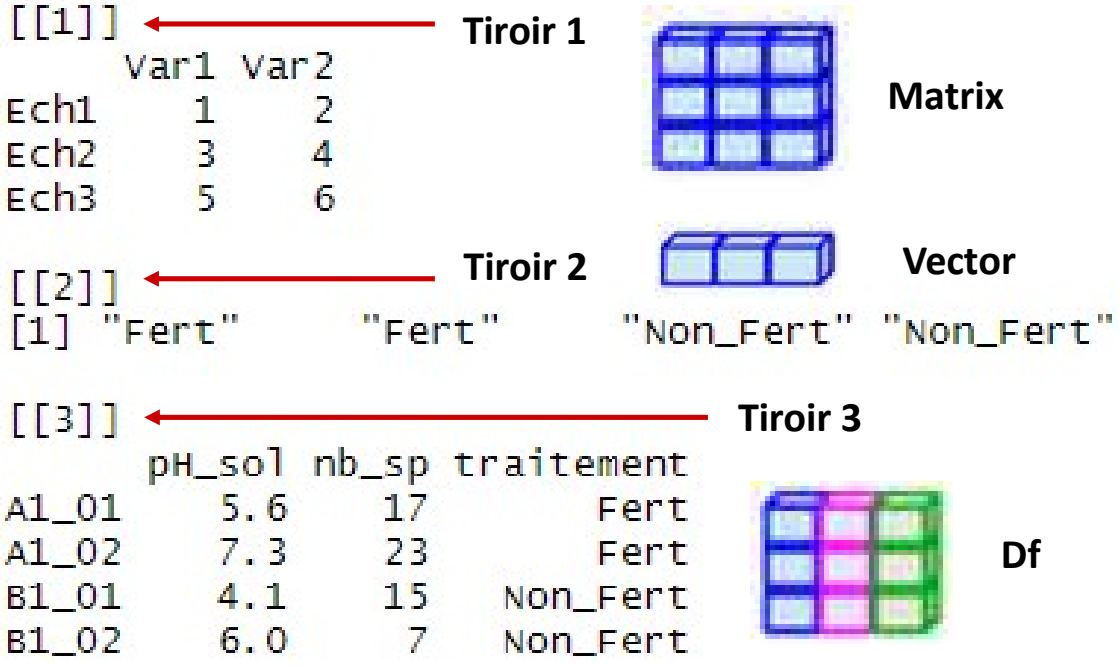
On peut considérer une liste comme une commode à plusieurs tiroirs permettant de stocker dans chacun d'eux des objets de structure différente.

Créer une liste avec la fonction list() :

```
mylist <- list (mat1, traitement, df)

mylist
```

Intérêt: regrouper dans une même structure des objets appartement à une même étude (expérience, observation) et de pouvoir les traiter en une seule fois grâce à des fonctions adaptées.

```
[[1]]
      Var1 Var2
Ech1     1    2
Ech2     3    4
Ech3     5    6

[[2]]
[1] "Fert"     "Fert"     "Non_Fert" "Non_Fert"

[[3]]
      pH_sol nb_sp traitement
A1_01    5.6    17       Fert
A1_02    7.3    23       Fert
B1_01    4.1    15   Non_Fert
B1_02    6.0     7   Non_Fert
```



8. Indexer un objet

Indexation : opération très utile permettant de sélectionner des sous-ensembles de valeurs en fonction de différents critères.

3 types :
- indexation par position/directe
- indexation par nom
- indexation par condition/logique.

Le principe : on indique entre crochets **[ ]** ce qu'on souhaite afficher/garder ou non.

S'applique aux différentes structures d'objets (vecteur, matrice, tableau, data frame, liste) avec certaines spécificités.

Permet également de remplacer des valeurs par d'autres.

8. Indexer un objet

Indexation sur vecteur : nom [n° élément]

par position/directe :

On veut la 2 valeur du vecteur

```
taille_tous_m [ 2 ]
```

La 2e et la 6e

```
taille_tous_m [ c (2 , 6) ]
```

Les valeurs de la position 3 à 5

```
taille_tous_m [ 3 : 5 ]
```

Toutes sauf la 4e et la 7e

```
taille_tous_m [ -c ( 4 , 7) ]
```

Exemple sur vecteur « taille_tous_m »

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	*1.56*	*1.64*	*1.97*	*1.47*	*1.73*	*1.88*

```
Sophie
  1.56
```

```
Sophie  Clara
  1.56   1.73
```

```
Julie Francois    Marie
 1.64     1.97     1.47
```

```
Marc Sophie  Julie  Marie  Clara
1.76   1.56   1.64   1.47   1.73
```

8. Indexer un objet

Indexation sur vecteur : nom [n° élément]

Exemple sur vecteur « taille_tous_m »

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	*1.56*	*1.64*	*1.97*	*1.47*	*1.73*	*1.88*

par nom :

On veut la taille de Sophie, Marie et Clara

```
taille_tous_m [ c ( "Sophie", "Marie", "Clara" ) ]
```

```
Sophie  Marie  Clara
  1.56   1.47   1.73
```

Autre option? … pas de – c () avec les étiquettes!

```
taille_tous_m [ -c ("Marc", "Julie", "Francois", "Theo")]
Error in -c("Marc", "Julie", "Francois", "Theo") :
  argument incorrect pour un opérateur unitaire
```

8. Indexer un objet

Indexation sur vecteur : nom [n° élément]

Exemple sur vecteur « taille_tous_m »

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	*1.56*	*1.64*	*1.97*	*1.47*	*1.73*	*1.88*

par condition/logique :

A l'aide d'un vecteur de booléens (T, F)

On veut la taille de Sophie, Marie et Clara

```
taille_tous_m [c (F, T, F, F, T, T, F ) ]
```

```
Sophie  Marie  Clara
  1.56   1.47   1.73
```

On veut connaître les éléments de taille >=1.78

```
taille_supeg178 <- taille_tous_m >= 1.78
```

```
  Marc  Sophie  Julie Francois  Marie  Clara  Theo
 FALSE   FALSE  FALSE     TRUE  FALSE  FALSE   TRUE
```

8. Indexer un objet

Indexation sur vecteur : nom [n° élément]

Exemple sur vecteur « taille_tous_m »

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	*1.56*	*1.64*	*1.97*	*1.47*	*1.73*	*1.88*

par condition/logique :

On veut sélectionner uniquement ces éléments

```
subset ( taille_tous_m, taille_supeg178==T )
subset ( taille_tous_m, taille_tous_m >=1.78 )
```

```
Francois      Theo
    1.97      1.88
```

Afficher le poids des individus de taille >=1.78

```
poids_tous [ taille_supeg178 ]
subset(poids_tous, taille_tous_m >=1.78)
```

```
Francois      Theo
      89       101
```

8. Indexer un objet

Indexation sur vecteur : nom [n° élément]

Exemple sur vecteur « taille_tous_m »

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	*1.56*	*1.64*	*1.97*	*1.47*	*1.73*	*1.88*

Pour modifier des valeurs :

On se rend compte que les valeurs pour Francois et Theo sont douteuses : NA (Not Available)

```
Taille_ex1 <- taille_tous_m
Taille_ex1 [ c (4, 7 ) ] <- NA
Taille_ex1
```

```
Marc   Sophie    Julie Francois   Marie   Clara    Theo
1.76    1.56     1.64       NA    1.47    1.73      NA
```

On remesure ces individus et on corrige :

```
Taille_ex2 <- taille_tous_m
Taille_ex2 [ c (4, 7) ] <- c (2, 1.9)
Taille_ex2
```

```
Marc   Sophie    Julie Francois   Marie   Clara    Theo
1.76    1.56     1.64     2.00    1.47    1.73    1.90
```

8. Indexer un objet

Indexation sur matrice ou data frame (2D) : nom [n° Ligne, n° Colonne]

Par position :

Ligne 1 entière :

```
mat1 [ 1 , ]
df [ 1 , ]
```

```
> mat1[1,]
Var1 Var2
   1    2
> df[1,]
      pH_sol nb_sp traitement
A1_01    5.6    17        Fert
```

Colonne 2 entière :

```
mat1 [ ,2 ]
df [ ,2 ]
```

```
> mat1[,2]
Ech1 Ech2 Ech3
   2    4    6
> df[,2]
[1] 17 23 15  7
```

Un élément précis :

```
mat1 [ 1, 2 ]
df [ 1, 2 ]
```

```
> mat1[1,2]
[1] 2
> df[1,2]
[1] 17
```

8. Indexer un objet

Indexation sur matrice ou data frame (2D) : **nom [n° Ligne, n° Colonne]**

Par position :

Un bloc de valeurs :

```
mat1 [ 2 : 3 , 1 : 2 ]
```

```
> mat1[2:3,1:2]
     Var1 Var2
Ech2    3    4
Ech3    5    6
```

```
df [ 2 : 3 , 1 : 3 ]
```

```
> df[2:3,1:3]
      pH_sol nb_sp traitement
A1_02    7.3    23       Fert
B1_01    4.1    15   Non_Fert
```

8. Indexer un objet

Indexation sur matrice ou data frame (2D) : **nom [n° Ligne, n° Colonne]**

Par nom : *(exemple sur data frame)*

```
df $ nb_sp [ 1 : 3 ]    # dataframe$nom_colonne
```

```
[1] 17 23 15
```

```
df [ , c ( "traitement" ,"nb_sp" ) ]
```

	traitement	nb_sp
A1_01	Fert	17
A1_02	Fert	23
B1_01	Non_Fert	15
B1_02	Non_Fert	7

par condition :

```
subset ( df, traitement=="Fert" & pH_sol>5 )
```

	pH_sol	nb_sp	traitement
A1_01	5.6	17	Fert
A1_02	7.3	23	Fert

```
subset (df, pH_sol>7, select = -c ( traitement ) )
```

	pH_sol	nb_sp
A1_02	7.3	23

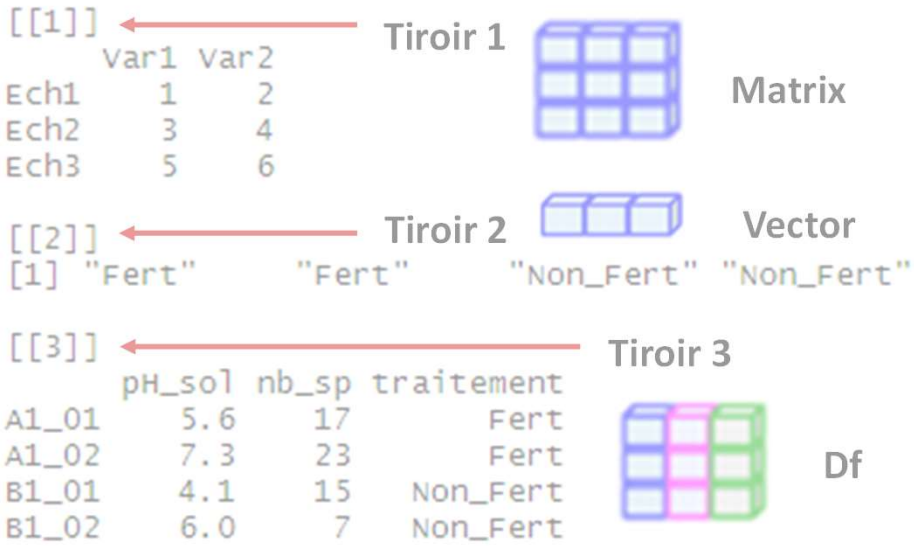
8. Indexer un objet

Indexation sur liste (>2D) : **nom [[n° objet]] [n° élement]**

Par position :

Ex pour le vecteur (2e objet) :

```
mylist [ [ 2 ] ]
mylist [ [ 2 ] ] [ 1 ]
```




```
> mylist[[2]]
[1] "Fert"     "Fert"     "Non_Fert" "Non_Fert"
> mylist[[2]][1]
[1] "Fert"
```

8. Indexer un objet

Indexation sur liste (>2D) : **nom [[n° objet]] [n° élement]**

Par nom :

En nommant les objets :

```
names ( mylist ) <-c ( "mat", "vec", "df" )
mylist
```

```
$mat
     Var1 Var2
Ech1    1    2
Ech2    3    4
Ech3    5    6

$vec
[1] "Fert"     "Fert"     "Non_Fert" "Non_Fert"

$df
      pH_sol nb_sp traitement
A1_01    5.6    17       Fert
A1_02    7.3    23       Fert
B1_01    4.1    15   Non_Fert
B1_02    6.0     7   Non_Fert
```

```
mylist $ df [ 1, 2 ]      #  liste $ element
```

```
[1] 17
```

9. Les fonctions

… Un autre concept de base de R …

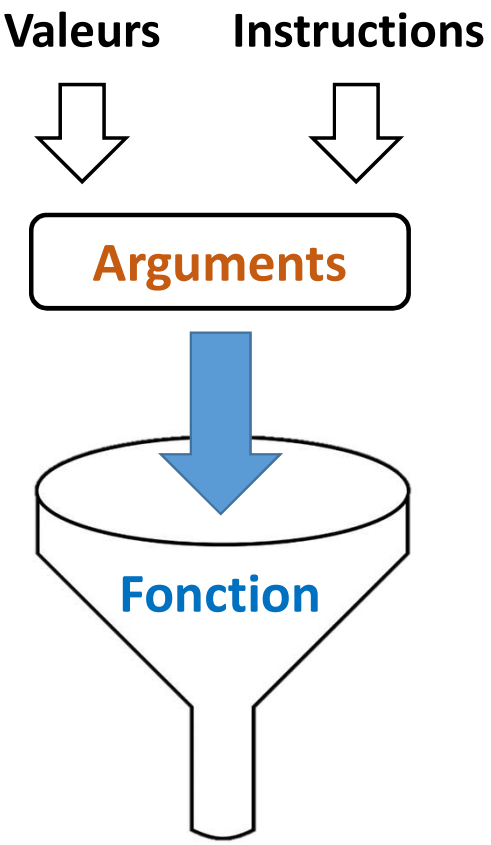
Fonction : Outil permettant d'exécuter des opérations sur des objets à partir de codes préprogrammés.

Caractéristiques :

- A un nom
- Nécessite valeurs d'entrées et instructions = **arguments** (peuvent avoir un nom ou pas)
- Retourne un **résultat** et/ou réalise une action (graphique…)

Pour appeler une fonction, la commande doit être structurée suivant les "règles de grammaire" du langage R (*i.e.* la **syntaxe**) :

***nom_de_la_fonction* (argument 1 , argument 2 , …, argument *i*)**



9. Les fonctions

Un exemple avec la fonction ”mean()” de calcul de la moyenne :

- Syntaxe : ***mean(x, trim =, na.rm =, ...)***
  - ***x*** : un objet contenant les valeurs à traiter (ex: vecteur)
  - ***trim*** : une valeur comprise entre 0 (par défaut) et 0.5 permettant de tronquer les deux extrémités de l’objet *x* avant le calcul de la moyenne
  - ***na.rm*** : un argument de type booléen TRUE ou FALSE (par défaut) permettant de retirer (TRUE) ou de conserver (FALSE) les valeurs manquantes dans *x*

NB: L'ordre des arguments est important → nécessaire de les nommer (avec signe =) s'ils ne sont pas appelés dans l'ordre.

Un argument obligatoire est un argument qui n’est pas suivi du signe = .

Un argument est optionnel s’il est suivi du signe = .

9. Les fonctions

Application de la fonction mean() sur des vecteurs précédemment crées :

```
mean(Taille_ex2)      # renvoie [1] 1.723   = moyenne sur l'ensemble des valeurs du vecteur

mean(Taille_ex1)      # renvoie [1] NA
```

… Pourquoi ?? …. (→ retour sur les données)

```
Taille_ex1
```

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
1.76	1.56	1.64	NA	1.47	1.73	NA

```
is.na(Taille_ex1)
```

Marc	Sophie	Julie	Francois	Marie	Clara	Theo
FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

```
which ( is.na ( Taille_ex1 ) )
```

Francois	Theo
4	7

```
mean(Taille_ex1, na.rm = T) # renvoie [1] 1.632
```

9. Les fonctions

6 fonctions utiles quand on travaille sur des data frames :

```
      pH_sol nb_sp traitement
A1_01    5.6    17       Fert
A1_02    7.3    23       Fert
B1_01    4.1    15   Non_Fert
B1_02    6.0     7   Non_Fert
```

df $ traitement <- **as.factor** (df $ traitement)
df $ traitement

```
[1] Fert     Fert     Non_Fert Non_Fert
Levels: Fert Non_Fert
```

str (df)

```
'data.frame':   4 obs. of  3 variables:
 $ pH_sol    : num  5.6 7.3 4.1 6
 $ nb_sp     : num  17 23 15 7
 $ traitement: Factor w/ 2 levels "Fert","Non_Fert": 1 1 2 2
```

summary (df)

```
     pH_sol          nb_sp          traitement
 Min.   :4.100   Min.   : 7.0   Fert    :2
 1st Qu.:5.225   1st Qu.:13.0   Non_Fert:2
 Median :5.800   Median :16.0
 Mean   :5.750   Mean   :15.5
 3rd Qu.:6.325   3rd Qu.:18.5
 Max.   :7.300   Max.   :23.0
```

9. Les fonctions

La fonction **apply()**, 3 arguments :

- Le data frame à manipuler
- FUN= la fonction à appliquer (mean, sum, sd…)
- MARGIN = 2 , fonction appliquée par colonne (numérique uniquement)

```
apply ( df [ , -c ( 3 ) ] , MARGIN = 2, FUN = mean )
```

```
pH_sol  nb_sp
  5.75  15.50
```

```
apply ( df [ , -c ( 3 ) ] , MARGIN = 2, FUN = sd )
```

```
   pH_sol    nb_sp
1.317826 6.608076
```

9. Les fonctions

Utiliser la fonction **ifelse()** pour recoder des variables :

```
df $ pH_inf6 <- ifelse ( df $ pH_sol <6, "inf6", "sup6" )
df $ pH_inf6 <- as.factor ( df $ pH_inf6 )
df
```

	pH_sol	nb_sp	traitement	pH_inf6
A1_01	5.6	17	Fert	inf6
A1_02	7.3	23	Fert	sup6
B1_01	4.1	15	Non_Fert	inf6
B1_02	6.0	7	Non_Fert	sup6

Agréger des données par variable catégorielle avec la fonction **aggregate()** :

```
aggregate ( df$nb_sp, by = df ["pH_inf6"], FUN = mean )
```

	pH_inf6	x
1	inf6	16
2	sup6	15

```
aggregate ( df$nb_sp, by = df ["traitement"], FUN = sum )
```

	traitement	x
1	Fert	40
2	Non_Fert	22

9. Les fonctions

[...] **Top 11 des fonctions utiles pour la manipulation de tableaux de données (dixit la promo précédente, UE Diagnostic) :**

- les fonctions dcast() et melt() du package reshape2
- les fonctions rbind()*, cbind()* et merge()
- la fonction row.names()*
- la fonction ifelse()*
- la fonction is.na() *
- les fonctions which()* et subset()*
- la fonction aggregate()*

*** = abordées dans ce TP** **[...]**

9. Les fonctions

Pour tracer un graphique de données :

- hist()
- pie()
- barplot()
- boxplot()
- plot()
- curve()
- dotchart()

Pour ajouter des éléments à un graphique existant :

- abline()
- points()
- lines()
- segments()
- polygon()
- title()
- legend()

[…]

10. HELP !!!

Préparez vous à quelques moments de frustrations intenses au départ !

10. HELP !!!

Erreur et avertissement

Warning message : Met en garde l'utilisateur sans arrêter l'exécution d'une fonction. Bien que la fonction donne une réponse, il pourrait y avoir un problème avec vos entrées → le calcul pourrait être erroné.

Error message : Arrête l'exécution en cours car R est incapable de réaliser le calcul.
→ Indique un problème dans votre code.

Pour résoudre une *erreur*, internet est votre meilleur ami!

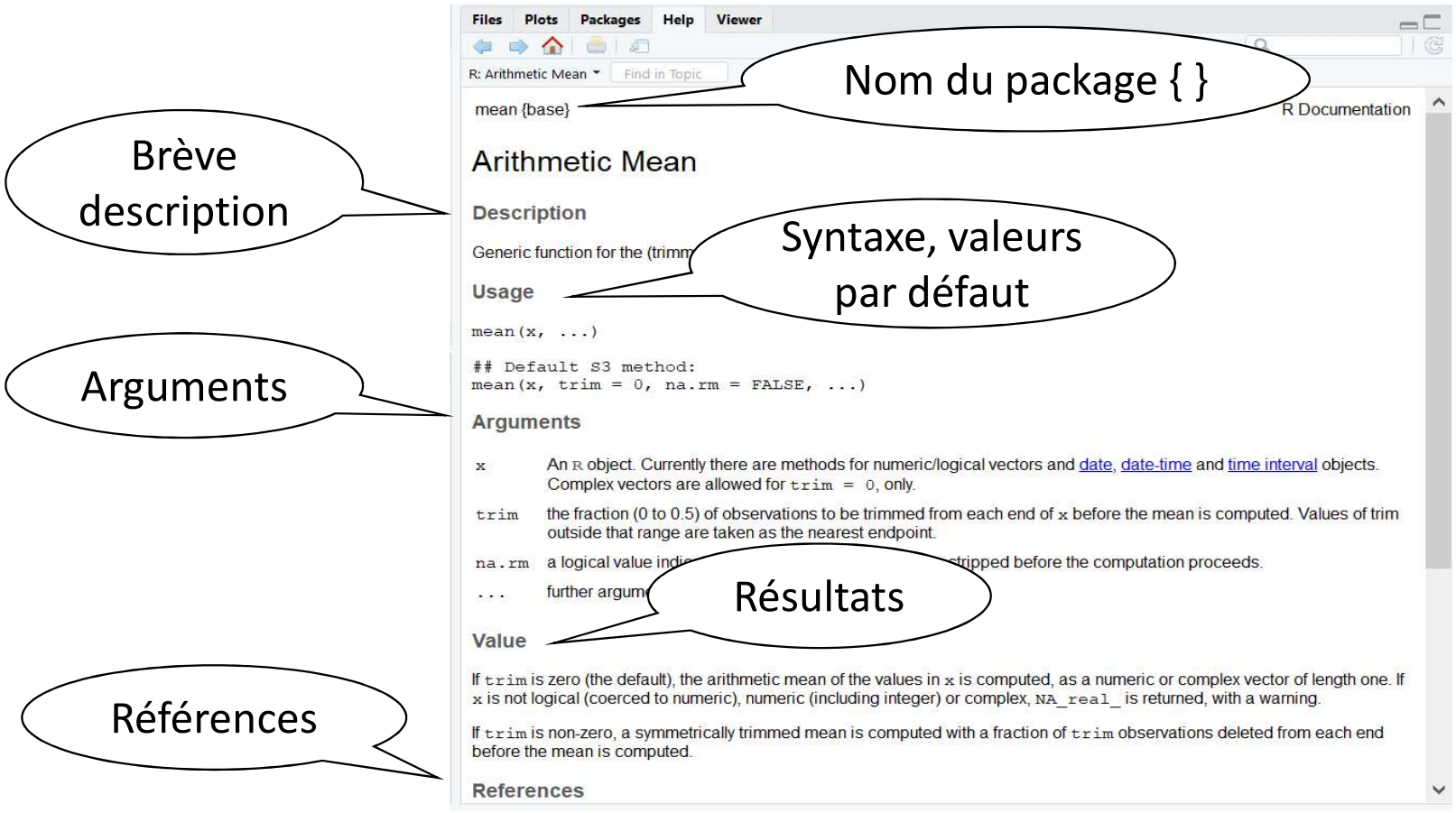
- Taper votre message d'erreur ou warning dans votre moteur de recherche…
- Faites vos recherches en anglais en commençant par le mot clé « R »
- Définissez précisément ce que vous cherchez, essayez des mots clés différents
- Apprenez à lire les discussions sur les forums, consultez des cours, livres tutos su R

La documentation officielle de R : http://www.r-project.org/

10. HELP !!!

Aide pour les fonctions : Comment ça marche? De quels arguments avez-vous besoin?

?nom_de_la_fonction # ex: **?mean**

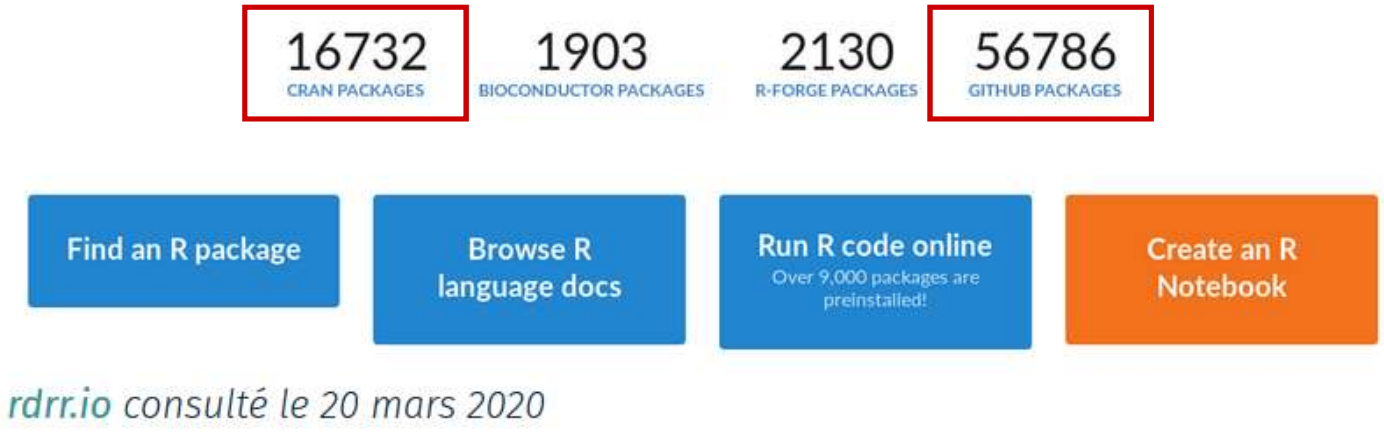


+ exemples reproductibles (...)

11. Vous avez dit « Package » ?

Les packages sont des regroupements de fonctions et/ou données partageant un thème similaire mis à disposition par la communauté (ex: analyses spatiales, graphiques…).

Beaucoup de packages sont disponibles via le *Comprehensive R Archive Network* (CRAN : http://cran.r-project.org/web/packages/), et maintenant encore plus via GitHub



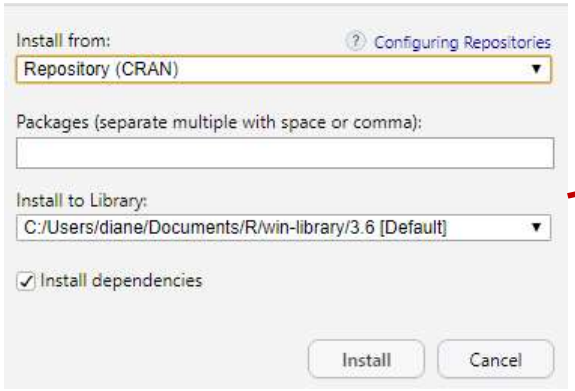
rdrr.io consulté le 20 mars 2020

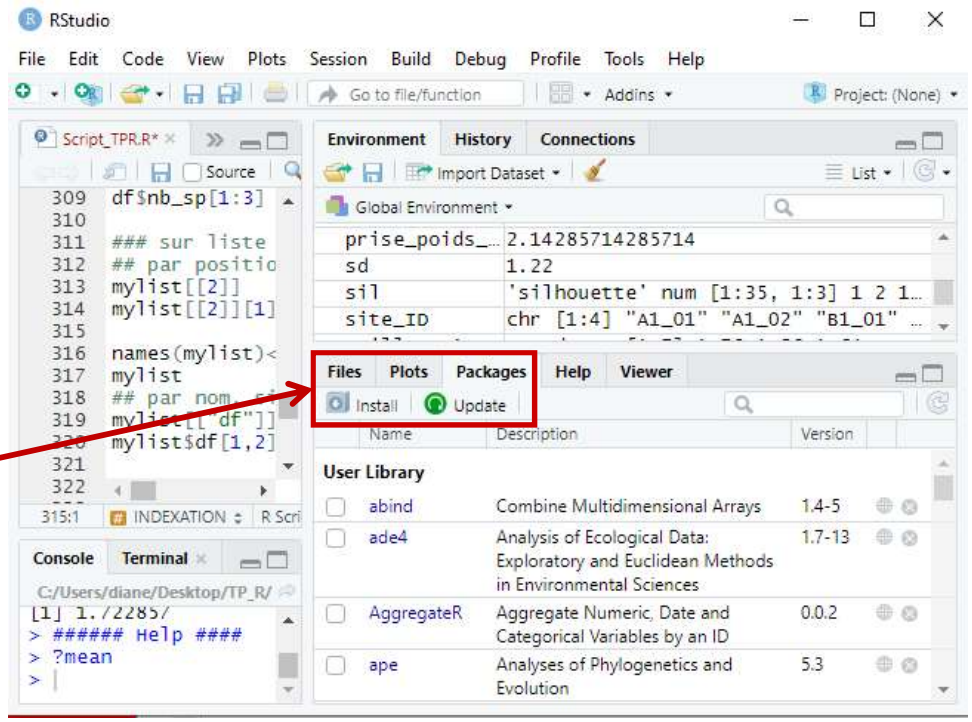
11. Vous avez dit « Package » ?

Installer un package :

install.packages (" nom_du_package ")

```
install.packages("ggplot2")
```





Il faut charger le package pour pouvoir l'utiliser:

library(nom_du_package)

```
library( ggplot2 )
```

Ressources :

Livres pdf:

https://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

https://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf

https://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.pdf

Super site avec tutos/cours pour manipuler et analyser ses données avec R :
https://larmarange.github.io/analyse-R/fusion-de-tables.html

Les webin-R associés :
https://www.youtube.com/channel/UCF9ymC7Dpjwr0lge2QR_A5Q

Exercice

1. Créez le tableau de données suivant, nommé « resultats » :

	taille	poids	QI	sexe
Paul	185	82	110	M
Matthieu	178	81	108	M
Camille	165	55	125	F
Mireille	171	65	99	F
Capucine	172	68	124	F

Exercice

2. Trouvez les lignes de code (commandes) permettant d'extraire les informations suivantes :

A. La taille de Camille

B. Le QI et le sexe des 3 premiers individus

C. Toutes les données relatives à Paul et Capucine

Exercice

3. Transformez la variable sexe en facteur et donnez la moyenne de taille par sexe.

4. Isolez dans un objet nommé res_M les résultats correspondant aux hommes uniquement.

5. En utilisant les opérateurs relationnels et logiques vus précédemment, sélectionnez les individus de sexe féminin avant un QI ≥ 100.

Exercice

TPR bonus pour aller plus loin…

(A faire chez vous, correction sur Moodle prochainement)

Allez sur Moodle et récupérez les fichiers suivants: