

Biologie et Modélisation - Introduction au logiciel



D. Chessel, A.B. Dufour, S. Penel, J. Lobry & N. Rochette





La fiche donne des indications pratiques très simples pour accéder au logiciel de statistique .

Table des matières

1	À propos de 	3
2	Avant de commencer	3
2.1	Créez un dossier de travail	3
2.2	Modifiez le raccourci de référence	3
2.3	Premiers pas	4
3	Les objets	4
3.1	Créez des objets : l'affectation	4
3.2	Les types d'objets ('classes') de base	6
4	Comment consulter l'aide de 	6
5	Utiliser l'historique	7
6	Vecteurs	7
6.1	 et le calcul sur les vecteurs	7
6.2	Création de vecteurs	8
6.3	Statistiques élémentaires	10
6.4	Mode d'un vecteur	10
6.5	Comparaisons et calculs logiques	11
6.6	La valeur singulière NA	12
6.7	Accès aux éléments : indexation	12
7	Matrices	14
7.1	Création de matrices	14
7.2	Indexation	15

8	Listes	15
8.1	La fonction <code>list()</code>	16
8.2	Accéder à un élément d'une liste	16
8.3	Ajout d'éléments	16
8.4	Sélection d'une partie d'une liste	17
9	Data.frames	17
9.1	Importer un data.frame depuis un fichier texte tabulé	17
9.2	Indexation : <code>[,]</code> et <code>\$</code>	19
9.3	Sélectionner des individus dans l'échantillon : <code>subset()</code>	21
10	Autres objets	21
10.1	Les fonctions	21
10.2	Les variables qualitatives : <code>factor</code>	21
10.3	Des objets pour tout	22

1 À propos de


R est un logiciel et un langage de programmation pour le calcul statistique. Il est distribué sous la licence libre GNU General Public License, dite licence GPL, via le réseau CRAN (Comprehensive R Archive Network) dont les sites de base sont :



<http://lib.stat.cmu.edu/R/CRAN/>
<http://stat.ethz.ch/CRAN/>

On peut utiliser le miroir de l'université :

<http://cran.univ-lyon1.fr/>

2 Avant de commencer

Le logiciel  est déjà installé dans les salles de TP. Néanmoins, il est recommandé d'effectuer les opérations qui suivent avant de commencer à travailler, sans quoi vous ne pourrez pas sauvegarder vos données.


(Ces instructions sont valables pour l'utilisation de  dans les salles de TP de l'université. Si vous voulez installer  sur votre machine personnelle et le personnaliser, consultez la fiche <http://pbil.univ-lyon1.fr/R/fichestd/tdr11.pdf>.)

2.1 Créez un dossier de travail

Sur le bureau, créez un nouveau dossier, ce sera notre dossier de travail pour ce cours. Évitez les espaces et les caractères spéciaux — ceci est vrai de manière générale en informatique ; pour les noms de dossiers et fichiers, mieux vaut se limiter aux caractères alphanumériques (lettres et chiffres), aux soulignés “_”, aux tirets et aux points.

Exemple : 'Biomod'.

2.2 Modifiez le raccourci de référence

Dans *Démarrer* → *Programmes* → *R*, faites un clic-droit sur le raccourci 'R 2.9.2', puis dans le menu pop-up *Envoyer vers* → *Bureau (créer un raccourci)*. Vérifiez qu'un raccourci  est apparu sur le bureau.

Déplacez ce raccourci vers votre dossier de travail et ouvrez le dossier.

Faites un clic-droit sur le raccourci, allez dans *Propriétés* et vérifiez que le champ *Démarrer dans* est vide. Sinon, videz-le et validez.

Avant de lancer le programme, vérifiez la configuration de l'explorateur de dossiers : dans *Outils* → *Option des dossiers* → *Affichage*, vérifiez :


- que le masquage des extensions de type connu est désactivé,
- que les fichiers cachés sont affichés,
- que *Afficher les chemins complets dans la barre d'adresses* est coché.

2.3 Premiers pas

Vous pouvez maintenant lancer le programme !

Dans la fenêtre principale du logiciel, vous remarquez en haut les barres de menus habituelles et au centre, une sous fenêtre 'R Console'. Nous allons principalement travailler dans cette fenêtre, aussi agrandissez-la.

Le plus important dans cette fenêtre est le chevron '>' que vous voyez à la dernière ligne. Ce symbole est *l'invite de commande* (*prompt* en anglais), qui marque l'endroit où l'on tape les commandes. Le texte qui le précède s'affiche chaque fois que vous démarrez une session et donne quelques informations diverses comme la version du logiciel, etc.

Dans  on travaille en lignes de commandes : le logiciel attend que l'on tape une commande (sur la ligne de l'invite de commande), répond à cette commande, fait apparaître un invite de commande à la suite de sa réponse, attend à nouveau, et ainsi de suite.

Avant de commencer à travailler, vérifions que les opérations précédentes se sont bien déroulées : placez vous dans la console, au bout de l'invite de commande, et entrez la commande suivante (envoyez avec entrée, ¶) :

```
getwd() ¶
```

Vous venez d'appeler la fonction `getwd()` ! Celle-ci affiche le **chemin absolu** du **répertoire courant** (ou dossier courant), c'est à dire le point de l'arborescence des fichiers et dossiers où le programme travaille à un instant donné. `getwd` veut dire 'get Working directory', 'récupérer le répertoire courant'.

Sous Windows, les chemins absolus commencent par 'C:/', 'D:/', etc. et sous Mac et Linux, simplement par '/'.


Une fois que l'on a défini un répertoire courant, on peut écrire des **chemins relatifs**. Tout chemin ne commençant pas par un 'X:/' (ou un '/') est un chemin relatif. Par exemple, le chemin 'pomme' désigne le dossier ou le fichier *pomme* dans le *répertoire courant*, si ce fichier existe.

Si la valeur retournée par la commande ne finit pas par Bureau/Biomod (ou le nom de votre dossier de travail, si vous avez choisi un nom différent), revenez au paragraphe précédent.


3 Les objets

Les données et les fonctions sont stockées dans des *objets*. Il existe de nombreux types d'objets, ou *classes*, destinés à contenir autant de types d'informations différents.


3.1 Créez des objets : l'affectation

Lorsque vous commencez votre session , certains objets sont déjà définis. C'est le cas de la fonction `getwd()` que nous avons utilisé tout à l'heure, de toutes les fonctions que nous utiliserons par la suite, et d'autres objets mathématiques ou statistiques courants comme le nombre `pi`.

pi ¶

Désormais, les commandes utilisées apparaîtront en rouge sur le cours, et le retour de  en bleu :

```
pi
[1] 3.141593
```

La première chose à faire pour créer/déclarer un objet est de lui trouver un nom. **Les noms d'objets  ne peuvent utiliser que les caractères alphanumériques (lettres minuscules et majuscules, chiffres), le souligné (underscore) et le point.** Les autres caractères sont interdits, le tiret en particulier car il est utilisé pour la soustraction.

Le nom d'un objet doit, dans la mesure du possible, donner des indications sur son contenu. Il s'agit ici de faire preuve de bon sens : on n'appelle pas 'v' un data.frame, ni 'alphabet' un vecteur de nombres !

Une fois que vous avez le nom de notre objet, créez-le en lui *assignant* une valeur. L'*opérateur d'assignation* est <-, composé des deux caractères '<' (chevron ouvrant) et '-' (tiret). Il s'utilise comme suit :

```
x <- 7
```

Voilà, nous avons créé un objet x contenant le nombre 7. On peut afficher le contenu de x en tapant simplement

```
x
[1] 7
```

Cela fonction de la même manière pour tous les objets. Ci-dessous, on crée trois objets contenant, respectivement, une valeur logique (un 'booléen'), un mot, et une liste vide.

```
b <- TRUE
mon_texte <- "toto"
l <- list()
```

Une fois les objets définis, on peut y faire appel à tout moment


```
mon_texte
[1] "toto"
b
[1] TRUE
```

La commande ls() donne les noms des objets que vous avez définis :

```
ls()
[1] "b"          "l"          "mon_texte" "x"
```

3.2 Les types d'objets ('classes') de base

Les classes basiques sont :

- **vector** Les vecteurs sont des séries ordonnées de nombres, de caractères (de mots) ou de valeurs logiques (vrai-faux, oui-non, ...). Quoi qu'il arrive, *tous les éléments d'un vecteur sont toujours de même type* : un vecteur de nombres ne contient que des nombres, un vecteur de mots ne contient que des mots, etc.
- **matrix** Les matrices sont semblables aux vecteurs, mais en deux dimensions (lignes, colonnes). Comme pour les vecteurs, tous les éléments sont du même type.
- **list** Les listes ressemblent aux vecteurs, en cela qu'elles contiennent des séries d'objets, mais il y a une très grosse différence : on peut mettre les objets que l'on veut dans une liste, et les mélanger. Une même liste peut contenir des vecteurs, une matrice, voire une autre liste : c'est une classe très flexible. De plus, la syntaxe pour les listes n'est pas la même que pour les vecteurs (voir le paragraphe sur les listes).
- **data.frame** : les **tableaux** Un tableau est une série de lignes partageant toutes les mêmes colonnes. Habituellement, chaque ligne correspond à un individu échantillonné et chaque colonne est un attribut (par exemple pour une étude démographique : age, sexe, couleur des yeux, etc.). Les tableaux sont le contenant naturel des données statistiques et nous les utiliserons abondamment au cours de l'UE.
- **function** Les fonctions sont elles aussi des objets. Elles contiennent du code  ; ce code est exécuté chaque fois que la fonction est appelée.

4 Comment consulter l'aide de

★ CONSULTER LES FICHES DE DOCUMENTATION ★
EST UNE OPÉRATION FONDAMENTALE !!

Pour obtenir de l'aide sur un objet on utilise le point d'interrogation ? :

`?objet`

Par exemple

`?getwd`

Vous pouvez aussi utiliser la commande `help(sujet)`.


Pour effectuer une recherche par mot clé, utilisez l'onglet "recherche" de la fenêtre d'aide.

Les fiches sont toutes structurées de la même manière. Outre le nom de la fiche et le module dont elle dépend (en haut à gauche), on retrouve les sections suivantes :

- Description : une courte description du contenu de la fiche
- Usage : la liste des arguments (paramètres) de la ou des fonctions présentées et de leurs valeurs par défaut
- Arguments : la description des arguments
- Details
- See also (optionnel) : des liens vers d'autres fiches sur des sujets connexes
- Examples

Examinez la fiche de la fonction `getwd()`. Vous remarquerez que la fiche décrit aussi la fonction `setwd()` (une fiche décrit souvent plusieurs fonctions sur le même sujet) ; essayez de comprendre ce que fait cette fonction.


5 Utiliser l'historique

 se souvient des commandes que vous avez tapées. Pour naviger dans l'historique des commandes, utilisez les flèches du haut et du bas.

Revenir à une commande tapée précédemment vous sera très utile pour corriger les erreurs dans une commande qui n'a pas fonctionné.

6 Vecteurs


6.1 et le calcul sur les vecteurs


 est conçu pour travailler avec des vecteurs, et propose une syntaxe très puissante pour leur manipulation.

Prenons la définition de la variance d'un échantillon de taille n d'une variable continue X :

$$\text{var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Dans un langage de programmation classique, il faudrait calculer la somme des éléments en les additionnant un à un avant de pouvoir calculer la moyenne \bar{x} , puis soustraire la moyenne à tous les éléments et élever la différence au carré, avant de pouvoir additionner les carrés, un à un toujours, et calculer enfin la variance.

Dans  pas besoin de boucles, on travaille directement sur tous les éléments de notre vecteur !

Exemple avec les dix premiers entiers. En  le vecteur des dix premiers entiers peut s'écrire `1:10` (voir paragraphe suivant). Le calcul de la variance se fait comme ceci :

```
x <- 1:10
x
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Longueur de x
n <- length(x)
# Calcul de la moyenne
m <- sum(x) / n
# Calcul de la variance
v <- sum( (x - m)^2 ) / n
v
[1] 8.25
```

Les expressions sont ainsi beaucoup plus compactes et proches des notations mathématiques.

1. Retrouvez la variance des 1000 premiers entiers :

```
[1] 83333.25
```

2. Retrouvez le carré des 10 premiers entiers moins un :

```
[1] 0 3 8 15 24 35 48 63 80 99
```

3. Retrouvez la somme (`sum()`) des 10 premiers entiers :

```
[1] 55
```

4. Retrouvez le sinus (`sin()`) des 10 premiers entiers :

```
[1] 0.8414710 0.9092974 0.1411200 -0.7568025 -0.9589243 -0.2794155 0.6569866
[8] 0.9893582 0.4121185 -0.5440211
```

5. Retrouvez le vecteur contenant 10 aux puissances des 10 premiers entiers :

```
[1] 1e+01 1e+02 1e+03 1e+04 1e+05 1e+06 1e+07 1e+08 1e+09 1e+10
```

6. Retrouvez le logarithme népérien (`log()`) du vecteur de 10 aux puissances des 10 premiers entiers :

```
[1] 2.302585 4.605170 6.907755 9.210340 11.512925 13.815511 16.118096 18.420681
[9] 20.723266 23.025851
```

7. Retrouvez la racine carrée (`sqrt()`) des 10 premiers entiers :

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
[10] 3.162278
```

8. Retrouvez la somme cumulée (`cumsum()`) des 10 premiers entiers :

```
[1] 1 3 6 10 15 21 28 36 45 55
```

9. Retrouvez le produit (`prod()`) des 10 premiers entiers :

```
[1] 3628800
```

10. Retrouvez les différences (`diff()`) des 10 premiers entiers avec leur précédent :

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

6.2 Création de vecteurs

c()

Le moyen le plus simple de créer un vecteur est d'utiliser la fonction combiner : `c()`

```
c(1.5, 7.2, pi)
[1] 1.500000 7.200000 3.141593
```

Exercice. Construire les vecteurs suivants :

```
[1] 11.1 2.7 3.3
[1] "Alpha" "Bravo" "Charlie" "Delta"
```

Double point

L'opérateur double point `:` permet de générer des séries d'entiers, ascendantes ou descendantes.

```
1:12
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

Exercice. Générer les vecteurs suivants :

```
[1] 8 9 10 11 12 13 14 15
[1] -3 -4 -5 -6 -7 -8 -9 -10
```

seq()

La fonction `seq()` permet de générer des séries de nombres équidistants :

```
seq(from = 1, to = 5)
[1] 1 2 3 4 5
seq(from = 1, to = 2, by = 0.1)
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
seq(from = 10, to = 50, length = 10)
[1] 10.00000 14.44444 18.88889 23.33333 27.77778 32.22222 36.66667 41.11111 45.55556
[10] 50.00000
```

Consultez la page d'aide de la fonction `seq()` puis entraînez-vous avec les exercices ci-dessous.

Exercice. Générer les séries suivantes :

```
[1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10
[1] 0.0 0.4 0.8 1.2 1.6 2.0
```

Remarque : on peut parfois simplifier l'écriture en utilisant le double point plutôt que `seq()`

```
seq(from=0, to=1, by=0.1)
# s'écrit aussi
0:10/10
```

rep()

La fonction `rep()` permet de construire des vecteurs ayant des éléments répétés.

```
?rep
rep(1:5, times=2)
[1] 1 2 3 4 5 1 2 3 4 5
rep(c("un", "deux"), times=c(3, 2))
[1] "un" "un" "un" "deux" "deux"
rep(1:5, length = 12)
[1] 1 2 3 4 5 1 2 3 4 5 1 2
rep(1:5, each = 2)
```

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

Consultez la page d'aide de la fonction `rep()` puis entraînez-vous avec les exercices ci-dessous.

Exercice. Générer les séries suivantes :

```
[1] 1 2 3 1 2 3 1 2 3
[1] 1 2 3 4 1 2 3 4 1 2 3 4
[1] 1 1 1 2 2 2 3 3 3 4 4 4
[1] "un" "un" "un" "deux" "deux" "deux" "deux" "deux"
```

À partir de fichiers

La création de vecteurs peut aussi se faire grâce aux fonctions `readLines()`, qui lit les lignes d'un fichier, et `scan()`, qui lit les mots d'un fichier. Nous n'aborderons l'utilisation de ces fonctions qu'au cours des prochaines séances.

6.3 Statistiques élémentaires

Considérons les notes de 14 étudiants d'un groupe de TP et calculons les paramètres statistiques élémentaires. Lire la documentation des fonctions pour comprendre ce qu'elles calculent.

```
notes <- c(15,8,14,12,14,10,18,15,9,5,12,13,12,16)
sort(notes)
length(notes)
min(notes)
max(notes)
range(notes)
median(notes)
quantile(notes)
mean(notes)
var(notes)
sd(notes)
unique(notes)
sort(unique(notes))
```

6.4 Mode d'un vecteur

Le mode d'un vecteur est le type de données qu'il contient. Les modes principaux sont `numeric` (pour les nombres réels), `logical` (pour les valeurs logiques vrai/oui et faux/non) et `character` (pour les chaînes de caractères – les "mots").

La fonction `mode()` permet de connaître le mode :

```
mode(1:10)
[1] "numeric"
```

Comme nous l'avons vu plus haut, *tous les éléments d'un vecteur sont toujours de même mode.*

Néanmoins, il peut arriver qu'un vecteur contienne des mots et des nombres ; mais alors *ces nombres ne sont pas vus comme des nombres (`numeric`) mais comme des mots (`character`)* et tous les éléments sont bien de mode `character`.

Exemple :

```
v <- c("alpha", "beta", TRUE, 4.3, "omega")
mode(v)
[1] "character"
v
[1] "alpha" "beta" "TRUE" "4.3" "omega"
```

Notez les guillemets autour de 'TRUE' et de '4.3' : il s'agit ici des chaînes de caractères TRUE et 4.3

6.5 Comparaisons et calculs logiques

Ce sont les calculs qui retournent une valeur logique vrai-faux.

Les opérateurs de comparaison sont '==' (égalité, avec *deux* signes égal), '>' (plus grand que), '>=' (plus grand ou égal), '<' et '<='.

Si deux vecteurs u et v de même longueur sont comparés, l'opération se fait deux à deux, c'est à dire que le premier élément de u est comparé avec le premier élément de v , le second au second, etc.

```
u <- 1:5
u
[1] 1 2 3 4 5
v <- 5:1
v
[1] 5 4 3 2 1
u == v
[1] FALSE FALSE TRUE FALSE FALSE
u <= v
[1] TRUE TRUE TRUE FALSE FALSE
u < v
[1] TRUE TRUE FALSE FALSE FALSE
```

L'opérateur égalité == peut aussi être utilisé sur les chaînes de caractères :

```
u <- c("Alpha", "Bravo", "Charlie", "toto")
u
[1] "Alpha" "Bravo" "Charlie" "toto"
v <- c("Alpha", "Beta", "Gamma", "toto")
v
[1] "Alpha" "Beta" "Gamma" "toto"
u == v
[1] TRUE FALSE FALSE TRUE
```

Les opérations logiques à proprement parler concernent la comparaison de valeurs logiques, c'est à dire le OU et le ET. L'expression "A OU B" est vraie si l'affirmation A *ou* l'affirmation B est vraie ; l'expression "A ET B" est vraie si A *et* B sont vraies.

Les opérateurs sont | (OU, AltGr+6) et & (ET) et comme pour les comparaisons numériques, elles se font deux à deux.

```
u <- c(FALSE, TRUE, TRUE)
v <- c(FALSE, FALSE, TRUE)
u | v
[1] FALSE TRUE TRUE
u & v
[1] FALSE FALSE TRUE
```

6.6 La valeur singulière NA

La valeur singulière NA signifie "Not Available" ("Non Disponible") et permet de composer au mieux avec les éventuelles données manquantes dans des données statistiques.

Elle se propage dans tous les calculs :

```
x <- c(1, 2, 3, 4, NA)
sum(x)
[1] NA
x <= 3
[1] TRUE TRUE TRUE FALSE NA
TRUE & NA
[1] NA
```

(Remarque : NA n'a pas de mode propre)

6.7 Accès aux éléments : indexation

L'accès aux éléments d'un vecteur se fait via l'opérateur []. Cet opérateur peut être utilisé de plusieurs manières :

Indexation par des entiers positifs

Soit x le vecteur :

```
x <- c(145, -1, 28.88, 0.02, 34.5)
x
[1] 145.00 -1.00 28.88 0.02 34.50
```

Le quatrième élément :

```
x[4]
[1] 0.02
```

Du second au troisième élément :

```
x[2:3]
[1] -1.00 28.88
```

On peut reprendre plusieurs fois le même :

```
x[c(2, 2, 3)]
[1] -1.00 -1.00 28.88
```

Les éléments hors bornes ne sont pas disponibles (NA, not available, donnée manquante) :

```
x[100]
[1] NA
```

Indexation par des entiers négatifs

Tous sauf le quatrième :

```
x[-4]
[1] 145.00 -1.00 28.88 34.50
```

Exercice 1. Donner tous les éléments sauf le premier.

```
[1] -1.00 28.88 0.02 34.50
```

Exercice 2. A l'aide de la fonction `length()`, donner tous les éléments sauf le dernier :

```
[1] 145.00 -1.00 28.88 0.02
```

On ne peut pas mélanger les indexations par des entiers positifs et négatifs simultanément. Il faut procéder en deux temps.

Exercice 3. En une seule commande, enlever le premier élément et donner les deuxième et troisième éléments du vecteur obtenu :

```
[1] 28.88 0.02
```

Indexation par un vecteur logique

```
x[c(T, F, F, T, T)]
[1] 145.00 0.02 34.50
x[c(T, F)]
[1] 145.00 28.88 34.50
```

Si le vecteur logique n'est pas assez long il sera recyclé autant de fois que nécessaire.

Exercice. Donner un élément sur deux à partir du deuxième :

```
[1] -1.00 0.02
```

Le vecteur logique d'indexation peut être issu d'un calcul logique, c'est l'utilisation la plus courante :

```
x > 10
[1] TRUE FALSE TRUE FALSE TRUE
x[x > 10]
[1] 145.00 28.88 34.50
x > 0 & x < 1
[1] FALSE FALSE FALSE TRUE FALSE
x[x > 0 & x < 1]
[1] 0.02
```

Exercice. Donner la liste des éléments compris entre 10 et 50 :

```
[1] 28.88 34.50
```

Indexation par des noms

Ceci ne fonctionne que si les éléments du vecteur ont un nom. Donnons comme nom les 5 premières lettres de l'alphabet aux éléments du vecteur `x`, en utilisant la fonction `names()` :

```
# le vecteur contenant l'alphabet est déjà défini !
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z"

# Les éléments de x n'ont pas (encore) de noms
names(x)
NULL

names(x) <- letters[1:5]
names(x)
[1] "a" "b" "c" "d" "e"
```

Quelles sont les valeurs des éléments `a` et `e` ?

```
x[c("a", "e")]
  a      e
145.0  34.5
```

On ne peut pas exclure des éléments par leur nom directement car l'opérateur d'exclusion `[-elements]` ne fonctionne qu'avec les positions des éléments (des entiers, donc). Il faut d'abord récupérer leurs indices avec la fonction `match()`.

Les positions des éléments `a` et `e` dans `x` sont :

```
match(c("a", "e"), names(x))
[1] 1 5
```

Exercices.

1. Quelles sont les valeurs des éléments autre que `a` et `e` ?

```
  b      c      d
-1.00 28.88  0.02
```

2. À partir de l'alphabet `letters`, quelles sont les consonnes ? Vous commencerez par définir un vecteur `voyelles`.

```
[1] "b" "c" "d" "f" "g" "h" "j" "k" "l" "m" "n" "p" "q" "r" "s" "t" "v" "w" "x" "z"
```

7 Matrices

7.1 Création de matrices

La création de matrices se fait avec la fonction `matrix()`. La syntaxe est `matrix(v, nrow=L, ncol=C, byrow=true/false)`, où `L` et `C` sont le nombre de lignes et de colonnes désirées, et `v` un vecteur contenant les éléments de la matrice – `v` doit être de longueur `Lx C`. Le paramètre `byrow` (français : 'par ligne') indique s'il faut remplir la matrice par ligne ou par colonne. Exemple :

```
m <- matrix(1:9, nrow = 3, ncol = 3, byrow = FALSE)
m
  [,1] [,2] [,3]
[1,]  1   4   7
[2,]  2   5   8
[3,]  3   6   9

m <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
m
```

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

Pour donner des noms aux lignes et aux colonnes :

```
rownames(m) <- c("a", "b", "c")
colnames(m) <- c("x", "y", "z")
m
  x y z
a 1 2 3
b 4 5 6
c 7 8 9
```

Exercice.

Créez la matrice de mode character qu'il vous plaira.

Alternativement, une matrice peut être créée depuis un fichier grâce à `scan()`.

7.2 Indexation

L'accès aux éléments d'une matrice utilise l'opérateur `[,]`. Cet opérateur fonctionne comme l'opérateur `[]` des vecteurs, sauf qu'on référence ici les deux dimensions de la matrice. Le premier champ correspond aux lignes, le second aux colonnes : `[1,2]` désigne l'élément au croisement de la 1^e ligne et de la 2^e colonne.

```
m[1, 2]
[1] 2
```


Comme avec les vecteurs on peut désigner plusieurs lignes ou colonnes à la fois. On peut aussi utiliser les noms des lignes et des colonnes, ou des vecteurs logiques.

```
m[ c(1,2) , c(FALSE,TRUE,TRUE) ]
  y z
a 2 3
b 5 6
```

Laisser un des champs de l'opérateur vide revient à sélectionner l'ensemble des éléments sur cette dimension : par exemple laisser le second champ vide sélectionne toutes les colonnes.

```
m[2, ]
  x y z
4 5 6
```

8 Listes

Les listes sont une structure de données très flexible et très utilisée dans . Un élément d'une liste est un objet R *quelconque*, y compris une autre liste.

8.1 La fonction `list()`

La fonction `list()` permet de créer des listes très simplement :

```
maliste <- list(a = pi, second = "une chaine", toto = c(T, F, NA))
maliste
$a
[1] 3.141593
$second
[1] "une chaine"
$toto
[1] TRUE FALSE NA
```

8.2 Accéder à un élément d'une liste

L'opérateur `[[]]`

Les listes sont *ordonnées*, et l'on accède aux éléments par leur position avec l'opérateur `[[]]`.

```
maliste[[1]]
[1] 3.141593
```

On peut aussi accéder un élément par son nom :

```
maliste[["toto"]]
[1] TRUE FALSE NA
```

Remarque : Contrairement aux opérateurs `[]` et `[,]`, l'opérateur `[[]]` n'accepte pas de vecteurs. Dans une liste, on extrait un seul élément à la fois.

L'opérateur `$`

Les éléments d'une liste sont le plus souvent nommés, or taper `[["nom"]]` est incommode. Il existe un raccourci d'écriture : l'opérateur `$`. L'on écrit simplement `$nom`, comme ceci :

```
maliste$toto
[1] TRUE FALSE NA
```

Exercice.

1. Donner les deux manières d'extraire l'élément "a" de la liste :

```
[1] 3.141593
```

2. Donner le deuxième élément de la liste :

```
[1] "une chaine"
```

8.3 Ajout d'éléments

Le plus simple est d'introduire un nouvel élément nommé :

```
maliste$d <- 1:10
```

8.4 Sélection d'une partie d'une liste


L'opérateur `[]` appliqué à une liste retourne une liste contenant *certaines* des éléments de la liste d'origine. Par exemple, si seuls les éléments "a" et "d" de notre liste `maliste` nous intéressent, on écrira :


```
autreliste <- maliste[c("a", "d")]
autreliste
$a
[1] 3.141593
$d
[1] 1 2 3 4 5 6 7 8 9 10
```

9 Data.frames

Un `data.frame` ou tableau est une série de lignes partageant toutes les mêmes colonnes. Des données d'échantillonnage trouvent leur place dans un tableau où chaque ligne est un individu, et les colonnes ses différents attributs.

9.1 Importer un `data.frame` depuis un fichier texte tabulé

Créer un tableau en lignes de commandes est assez fastidieux. Plusieurs solutions s'offrent à vous : utiliser un éditeur de tableaux, si votre version de  en est munie (ce n'est pas le cas dans les salles de TP), ou importer votre tableau depuis un fichier texte tabulé grâce à la commande `read.table()`.

Pour ce TP, nous allons écrire notre tableau dans Excel, le sauvegarder au format texte tabulé, puis l'importer dans .

1. Créer un fichier Excel en tapant sur trois colonnes (`sex`, `poi`, `taï`) les données ci-dessous. (Contentez vous des **10 premières lignes** seulement.)

	A	B	C
1	sexe	poi	taï
2	h	60	170
3	f	57	169
4	f	51	172
5	f	55	174
6	f	50	168
7	f	50	161
8	f	48	162
9	h	72	189
10	f	52	160
11	h	64	175
12	f	53	165
13	h	72	164
14	h	61	175
15	h	78	184
16	h	68	178
17	f	51	158
18	f	53	164
19	h	79	179
20	h	74	182
21	h	62	174
22	f	49	158

	A	B	C
23	f	50	163
24	h	74	172
25	h	80	185
26	f	53	170
27	h	73	178
28	h	70	180
29	h	72	189
30	f	70	172
31	f	62	174
32	h	77	200
33	h	70	178
34	h	76	178
35	f	51	168
36	f	52	170
37	f	57	160
38	f	53	163
39	f	55	168
40	f	66	172
41	h	65	175
42	h	75	180
43	f	50	162
44	f	53	177

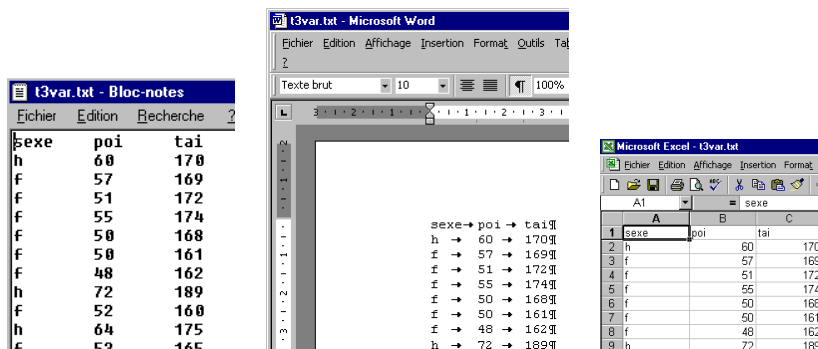
	A	B	C
45	h	55	169
46	h	55	173
47	h	72	182
48	h	75	183
49	h	73	184
50	h	71	181
51	h	66	180
52	h	71	178
53	h	79	178
54	h	62	168
55	f	47	161
56	h	73	171
57	h	72	180
58	h	60	174
59	h	67	175
60	h	85	182
61	h	73	181
62	h	82	188
63	h	86	182
64	h	85	189
65	h	65	178
66	f	47	150
67	h	74	186


2. Quand vous avez terminé, sauvegardez-le au format Excel dans le fichier `t3var.xls`, puis utiliser la commande `enregistrer sous ...` pour sauvegarder cette information au format Texte(séparateur: tabulation) :

Nom du fichier : t3var.txt
 Type de fichier : Texte (séparateur: tabulation) (*.txt) (*.txt)

(Excel vous demandera de confirmer le format, répondez oui.)

- Ouvrir le fichier créé, qui doit être dans votre dossier de travail, par un double clic pour le voir dans l'éditeur de texte, puis par *Word* pour afficher les caractères cachés, puis l'ouvrir avec *Excel* pour bien comprendre que la même information est lue et utilisée par différents programmes. Ouvrez aussi le fichier .xls dans un éditeur de texte (*Clic droit* → *Ouvrir avec* → *Bloc notes*).



- Une fois votre tableau sauvegardé dans un fichier texte, revenez dans 


Avant d'utiliser la fonction `read.table()` pour importer votre tableau, consultez sa fiche de documentation. À quoi servent les arguments `file`, `header` et `sep`?

`?read.table`

Observons d'abord ce qui se passe quand on lit les 5 premières lignes du tableau.

```
read.table("t3var.txt")[1:5, ]
  V1 V2 V3
1 sexe poi tai
2 h 60 170
3 f 57 169
4 f 51 172
5 f 55 174
```

Le premier individu contient les en-têtes ! Donnez la bonne valeur au paramètre `header`. Quand le résultat semble correct, placer le résultat de cette lecture dans un objet "t3var".

Dans le cas précédent, le fichier à charger dans  était sauvegardé dans un répertoire local. Il est également possible de charger des données sauvegardées sur un répertoire distant et notamment sur un site internet. Un grand nombre de jeux de données sont disponibles sur le site

<http://pbil.univ-lyon1.fr/R/donnees/>

Visiter ce dossier, repérer le fichier `t3var`, l'importer à la main dans son dossier de travail, l'afficher dans le navigateur, l'ouvrir avec un éditeur, le télécharger directement dans son dossier. On peut même le lire directement par :

```
t3var <- read.table("http://pbil.univ-lyon1.fr/R/donnees/t3var.txt",
  h = T)
```

9.2 Indexation : [,] et \$

Les colonnes (attributs) sont directement accessibles par leur nom :

```
t3var$toi
[1] 170 169 172 174 168 161 162 189 160 175 165 164 175 184 178 158 164 179 182 174
[21] 158 163 172 185 170 178 180 189 172 174 200 178 178 168 170 160 163 168 172 175
[41] 180 162 177 169 173 182 183 184 181 180 178 178 168 161 171 180 174 175 182 181
[61] 188 182 189 178 150 186
```

Exercice.

1. Donner la moyenne des tailles :

```
[1] 174.0606
```

2. Donner la médiane des poids :

```
[1] 65.5
```

3. Donner la variance des poids :

```
[1] 123.5767
```

4. Donner l'écart-type des poids :

```
[1] 11.11651
```

On peut également utiliser l'opérateur `[,]` que l'on a déjà vu pour les matrices. Par exemple, pour avoir les 5 premiers individus :

```
t3var[1:5, ]
  sexe poi tai
1    h  60 170
2    f  57 169
3    f  51 172
4    f  55 174
5    f  50 168
```

Pour avoir les première et troisième colonnes des 5 premiers individus :

```
t3var[1:5, c(1, 3)]
  sexe tai
1    h 170
2    f 169
3    f 172
4    f 174
5    f 168
```

Tous les types d'indexation vu pour les vecteurs (entiers positifs, négatifs, logiques, noms) sont utilisables.

Exercice.

1. Donner les individus 1, 5 et 55 :

```
  sexe poi tai
1    h  60 170
5    f  50 168
55   h  73 171
```

2. Pour les 10 premiers individus, donner toutes les variables sauf la première :

```

      poi tai
1    60 170
2    57 169
3    51 172
4    55 174
5    50 168
6    50 161
7    48 162
8    72 189
9    52 160
10   64 175

```

3. Donner tous les individus de sexe féminin :

```

      sexe poi tai
2      f  57 169
3      f  51 172
4      f  55 174
5      f  50 168
6      f  50 161
7      f  48 162
9      f  52 160
11     f  53 165
16     f  51 158
17     f  53 164
21     f  49 158
22     f  50 163
25     f  53 170
29     f  70 172
30     f  62 174
34     f  51 168
35     f  52 170
36     f  57 160
37     f  53 163
38     f  55 168
39     f  66 172
42     f  50 162
43     f  53 177
54     f  47 161
65     f  47 150

```

4. Donner tous les individus qui font plus de 175 cm :

```

      sexe poi tai
8      h  72 189
14     h  78 184
15     h  68 178
18     h  79 179
19     h  74 182
24     h  80 185
26     h  73 178
27     h  70 180
28     h  72 189
31     h  77 200
32     h  70 178
33     h  76 178
41     h  75 180
43     f  53 177
46     h  72 182
47     h  75 183
48     h  73 184
49     h  71 181
50     h  66 180
51     h  71 178
52     h  79 178
56     h  72 180
59     h  85 182
60     h  73 181
61     h  82 188
62     h  86 182
63     h  85 189
64     h  65 178
66     h  74 186

```

5. Donner tous les individus de sexe féminin qui font plus de 175 cm :

```

      sexe poi tai
43     f  53 177

```

6. Donner le poids et la taille tous les individus de sexe féminin qui font plus de 175 cm :

```
poi tai
43 53 177
```

9.3 Sélectionner des individus dans l'échantillon : subset()

Il est très courant de sélectionner les individus (en ligne) en fonction de critères calculés sur les variables (en colonne). Pour éviter d'explicitier à chaque fois le nom du `data.frame` pour désigner une variable, on peut avantageusement utiliser la fonction `subset()`. Ainsi, la sélection des individus de sexe féminin qui font plus de 175 cm s'écrit aussi :

```
subset(t3var, sexe == "f" & tai > 175)
sexe poi tai
43 f 53 177
```

10 Autres objets

10.1 Les fonctions

Les fonctions sont aussi des objets! Tout comme on peut créer des vecteurs ou des `data.frames`, on peut écrire de nouvelles fonctions – c'est même très courant.

La syntaxe est la suivante :

```
nom <- fonction(arguments) {
  ... code de la fonction ...
  return(valeur)
}
```

(Notez qu'en anglais, fonction s'écrit avec un 'u'!)

Exemple :

```
mafonction <- fonction(x) {
  y <- x^2
  return(y)
}
mafonction(x=3)
[1] 9
mafonction(x=5)
[1] 25
```

Remarque : ici, `x` n'est pas un objet, mais simplement le nom d'un paramètre.

10.2 Les variables qualitatives : factor

Les facteurs sont la représentation sous \mathbb{R} des *variables qualitatives* (la couleur des yeux, le genre (σ , φ), un niveau de douleur, etc). On peut voir les facteurs comme des vecteurs pour lesquels on aurait précisé qu'ils décrivent une variable qualitative. Cette précision permet d'utiliser les fonctions applicables seulement aux variables qualitatives (tables de contingence, etc.).

```
douleur <- c(0, 3, 2, 2, 1)
fdouleur <- factor(douleur, levels = 0:3)
is.numeric(fdouleur)
[1] FALSE
is.character(fdouleur)
[1] FALSE
is.factor(fdouleur)
[1] TRUE
summary(fdouleur)
0 1 2 3
1 1 2 1
table(fdouleur)
fdouleur
0 1 2 3
1 1 2 1
```

La fonction `levels()` permet de récupérer ou de modifier les modalités des variables qualitatives, la fonction `as.numeric()` donne le codage numérique des modalités :

```
levels(fdouleur)
[1] "0" "1" "2" "3"
levels(fdouleur) <- c("rien", "leger", "moyen", "fort")
fdouleur
[1] rien fort moyen moyen leger
Levels: rien leger moyen fort
levels(fdouleur)
[1] "rien" "leger" "moyen" "fort"
as.numeric(fdouleur)
[1] 1 4 3 3 2
```

10.3 Des objets pour tout

Il existe en réalité une multitude d'objets pour représenter toutes sortes d'informations statistiques. Les lister toutes n'est pas le but de ce cours, mais il faut être conscient de l'existence de cette diversité. À titre d'exemple, il existe une classe `histogram` (pour les histogrammes!) et une classe `table` (pour les tables de contingence – voir vos cours de statistiques).

À tout moment on peut connaître la classe d'un objet grâce à la fonction `class()`. (Les vecteurs n'étant pas une classe à proprement parler, cette fonction retournera le type de données contenu, soit le *mode* du vecteur.)

FIN DU PREMIER TP.