

## Aide

La plupart des fonctions de R ont une documentation en ligne

**help(sujet)** documentation sur un **sujet**. Flèches haut et bas pour se déplacer, touche q pour quitter  
**?topic** idem  
**help.search("sujet")** recherche dans l'aide  
**apropos(" sujet")** le nom de tous les objets dans la liste de recherche qui correspondent à l'expression régulière « sujet »  
**help.start()** démarre la version HTML de l'aide (indispensable; le moteur de recherche intégré nécessite Java installé sur votre ordinateur)  
**example(function)** exécute l'exemple donné en bas de la page d'aide de la **function** indiquée

## Fonctions de base

**<-** et **->** assignation dans le sens de la flèche (**a <- b** équivaut donc à **b->a**); e.g.: **x<-0**; **x+1->x** (met x à 0, puis additionne x et 1 pour mettre le résultat dans x)  
**NULL** l'ensemble vide  
**NA** valeur manquante (Not Available)  
**"abc"** une chaîne de 3 caractères  
**str(a)** affiche la **structure** d'un objet R  
**summary(a)** donne un « résumé » de **a**, généralement un résumé statistique, mais c'est une fonction générique (fonctionne différemment selon la classe de **a**)  
**ls()** liste les objets de la liste de recherche; spécifier e.g. **pat="MonTexte"** pour chercher selon un patron  
**ls.str()** **str()** pour chaque variable de la liste de recherche  
**dir()** lister les fichiers dans le dossier (**directory**) en cours  
**methods(a)** afficher les méthodes S3 de **a**  
**methods(class=class(a))** lister toutes les méthodes permettant de traiter les objets de la classe de l'objet **a**  
**options(...)** définit ou examine de nombreuses options globales; options fréquentes : **width** (largeur en nombre de caractères de la fenêtre des résultats), **digits** (nombre de chiffres significatifs à l'affichage), **error** (traitement des erreurs)  
**library(x)** charge des packages additionnels;  
**library(help=x)** liste les jeux de données et fonctions du package **x**.  
**attach(x)** ajoute le contenu de **x** dans la liste de recherche de R; **x** peut être une liste, une data frame, ou un fichier de données R créé avec **save**. Utilisez **search()** pour montrer la liste de recherche.  
**detach(x)** enlève le contenu de **x** de la liste de recherche de R; **x** peut être un nom ou une chaîne de caractères désignant un objet préalablement attaché ou un package.  
**q()** quitter R (répondre **y** (yes) et Entrée pour confirmer)

## Entrée et sortie

**load()** charge le jeu de données écrit avec **save**  
**data(x)** charge le jeu de données spécifié  
**read.table(file)** lit un fichier au format tabulaire et en fait un data frame; le séparateur de colonne par défaut **sep=""** désigne n'importe quel espacement; utilisez **header=TRUE** pour prendre la première ligne comme titre (header) de colonne; utilisez **as.is=TRUE** pour empêcher les vecteurs de caractères d'être transformés en **factors**; utilisez **skip=n** pour ignorer les **n** premières lignes ; consultez l'aide pour les options concernant le nommage des colonnes, le traitement des valeurs manquantes (NA), etc.  
**read.csv2("filename",header=TRUE)** idem mais avec des options pré-définies pour lire les fichiers CSV

**read.delim("filename",header=TRUE)** idem mais avec des options pré-définies pour lire les fichiers dont les valeurs sont séparées par des tabulations  
**read.fwf(file,widths,header=FALSE,sep=" ",as.is=FALSE)** lit un tableau dont toutes les colonnes ont la même largeur (fwf: **fixed width format**); **widths** est un vecteur d'entiers donnant la largeur des colonnes dans le fichier  
**save("fichier", x,y)** enregistre les objets x et y dans le **fichier**, au format binaire XDR propre à R  
**save.image("fichier")** enregistre tous les objets  
**cat(..., file="", sep=" ")** affiche les arguments après les avoir converti en caractères; **sep** est le séparateur entre les arguments  
**print(a, ...)** affiche les arguments; fonction générique (fonctionne différemment selon la classe de **a**)  
**format(x,...)** formate un objet R pour un affichage personnalisé  
**write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ")** affiche **x** après l'avoir converti en data frame; si **quote** est **TRUE**, les colonnes de caractères ou de **factors** sont entourés par des guillemets; **sep** est le séparateur de colonnes. Avec **file=** suivi du chemin d'un fichier, écrit sur le disque dur

La plupart des fonctions d'entrée/sortie ont un argument **file**.

Cela peut souvent être une chaîne de caractères nommant un fichier ou une connexion. Sous Windows, la connexion peut aussi être utilisée avec **description="clipboard"** pour lire un tableau copié d'un tableau par le presse-papier

**x <- read.delim("clipboard")** pour lire un tableau copié par le presse-papier depuis un tableau  
**write.table(x,"clipboard",sep="\t",col.names=NA)** pour écrire un tableau vers le presse-papier pour un tableau

## Création de données

**c(...)** fonction combinant les arguments pour former un vecteur; avec **recursive=TRUE** va dans les listes pour combiner leurs éléments en un seul vecteur (plutôt qu'en un vecteur de listes)  
**de:vers** génère une séquence d'entiers; ":" est prioritaire: **1:4 + 1** vaut "2,3,4,5"  
**seq(from,to)** génère une séquence; **by=** spécifie l'incrément; **length=** spécifie la longueur  
**seq(along=x)** génère une suite **1, 2, ..., length(x)**; utile pour les boucles **for**  
**rep(x,times)** répète **times** fois la valeur **x**; utilisez **each=n** pour répéter **n** fois chaque élément de **x**;  
**rep(c(1,2,3),2)** vaut 1 2 3 1 2 3;  
**rep(c(1,2,3),each=2)** vaut 1 1 2 2 3 3  
**data.frame(...)** crée un data frame avec les arguments (nommés ou non); e.g: **data.frame(v=1:4, ch=c("a", "b", "c", "d"), lettre="A")**; les vecteurs plus courts (ici: **"A"**) sont réutilisés (recyclés) plusieurs fois pour atteindre la longueur du vecteur le plus long ; à la différence d'une **matrix**, un **data.frame** est un tableau dont les colonnes peuvent être de types différents  
**list(...)** crée une liste avec les arguments (nommés ou non, qui peuvent être de longueur différente); e.g.: **list(a=c(1,2),b="hi",c=3)**;  
**matrix(x,nrow=,ncol=)** crée une matrice (tous les éléments sont de même type); les éléments se répètent s'ils sont trop courts

**factor(x, levels=)** transforme un vecteur **x** en **factor** (les niveaux sont indiqués par **levels=**)  
**rbind(...)** combine les arguments par ligne (*row*)  
**cbind(...)** combine les arguments par colonne

### Extraction de données

#### *Indexer des listes*

**x[i]** le ou les éléments **i** de la liste (renvoyé(s) sous forme de liste, à la différence des cas suivants; fonctionne comme pour les vecteurs)  
**x[[n]]**  $n^{\text{ième}}$  élément de la liste  
**x[["nom"]]** l'élément nommé "nom"  
**x\$nom** l'élément nommé "nom"

#### *Indexer des vecteurs*

**x[n]**  $n^{\text{ième}}$  élément  
**x[-n]** tous sauf le  $n^{\text{ième}}$  élément  
**x[1:n]** les  $n$  premiers éléments  
**x[-(1:n)]** les éléments de  $n+1$  à la fin  
**x[c(1,4,2)]** des éléments spécifiques  
**x["nom"]** l'élément nommé "nom"  
**x[x > 3]** tous les éléments plus grands que 3  
**x[x > 3 & x < 5]** tous les éléments plus grands que 3 et plus petits que 5  
**x[x %in% c("a", "and", "the")]** les éléments appartenant à l'ensemble donné

#### *Indexer des matrices*

**x[i, j]** l'élément de la ligne **i**, colonne **j**  
**x[i, ]** toute la ligne **i**  
**x[, j]** toute la colonne **j**  
**x[, c(1,3)]** les colonnes 1 et 3  
**x["nom", ]** la ligne nommée "nom"

#### *Indexer des data.frame (comme pour les matrices plus ce qui suit)*

**x[["nom"]]** la colonne nommée "nom"  
**x\$nom** la colonne nommée "nom"

#### *Conversion d'objets*

**as.data.frame(x), as.numeric(x), as.logical(x), as.character(x), ...**  
conversion de type, e.g.: **as.logical(x)** convertit **x** en **TRUE** ou **FALSE**; pour la liste complète, faites **methods(as)**

#### *Information sur les objets*

**is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ...** tests de type; renvoie **TRUE** ou **FALSE**; pour une liste complète, faites **methods(is)**  
**length(x)** nombre d'éléments dans **x**  
**dim(x)** récupère ou définit (**dim(x) <- c(3,2)**) les dimensions d'un objet  
**nrow(x)** et **NROW(x)** nombre de lignes; **NROW(x)** considère un vecteur comme une matrice  
**ncol(x)** et **NCOL(x)** idem pour les colonnes  
**class(x)** récupère ou définit la classe de **x**; **class(x) <- "maclasse"**  
**unclass(x)** enlève l'attribut de classe de **x**  
**attr(x, which=)** récupère ou définit un attribut de **x**  
**attributes(x)** récupère ou définit la liste des attributs de **x**  
**which.max(x)** trouve l'indice du plus grand élément de **x**  
**which.min(x)** trouve l'indice du plus petit élément de **x**  
**rev(x)** renverse l'ordre des éléments de **x**  
**sort(x)** trie les éléments de **x** par ordre croissant; pour l'ordre décroissant: **rev(sort(x))**

**order()** renvoie une série d'indices permettant de permuter un tableau afin de le mettre dans l'ordre selon les valeurs de certaines colonnes; e.g., trier par ordre alphabétique de prénom le tableau suivant: **x <- data.frame(prenom=c("Bernard", "Charles", "Annie"), age=c(10,20,30)); x[order(x\$prenom), ]**  
**cut(x, breaks)** découpe **x** en intervalles (**factors**); **breaks** est le nombre de cas ou un vecteur de cloisons  
**which(x == a)** renvoie les indices de **x** pour lesquels le résultat de l'opération logique est vrai (**TRUE**), dans cet exemple les valeurs de **i** pour lesquelles **x[i]==a** (l'argument de cette fonction doit être une variable de type « logique » (vrai ou faux))  
**na.omit(x)** supprime les observations avec des valeurs manquantes (**NA**: *not available*); supprime les lignes correspondantes si **x** est une matrice ou un data.frame)  
**unique(x)** renvoie **x** sans les éléments dupliqués (pour un data.frame, ne renvoie que des lignes uniques)  
**table(x)** renvoie une table avec le décompte de chaque valeur différente de **x**; **table(x,y)** renvoie un tableau de contingence

#### *Mathématiques*

**sin, cos, tan, log, log10, exp** fonctions mathématiques  
**max(x)** maximum des éléments de **x**  
**min(x)** minimum des éléments de **x**  
**range(x)** mini et maxi: **c(min(x), max(x))**  
**sum(x)** somme des éléments de **x**  
**diff(x)** différence entre chaque élément de **x** et son prédécesseur  
**prod(x)** produit des éléments de **x**  
**mean(x)** moyenne des éléments de **x**  
**median(x)** médiane des éléments de **x**  
**quantile(x, probs=)** quantiles correspondant aux probabilités données; le paramètre par défaut **probs=c(0, .25, .5, .75, 1)** donne les quartiles  
**weighted.mean(x, w)** moyenne pondérée de **x** (pondération par **w**)  
**rank(x)** rang des éléments de **x**  
**var(x)** OU **cov(x)** variance des éléments de **x** (calculé avec  $n-1$  au dénominateur); si **x** est une matrice ou un data.frame, la matrice de variance-covariance est calculée  
**sd(x)** écart-type (*standard deviation*) de **x**  
**cor(x)** matrice de corrélation de **x** (pour une matrice ou un data.frame)  
**var(x, y)** OU **cov(x, y)** covariance entre **x** et **y**, ou entre les colonnes de **x** et celles de **y** si ce sont des matrices ou des data.frames.  
**cor(x, y)** coefficient de corrélation linéaire entre **x** et **y**, ou matrice de corrélation si ce sont des matrices ou des data.frames.  
**round(x, n)** arrondit les éléments de **x** à **n** décimales  
**pmin(x,y,...)** un vecteur dont le  $i^{\text{ème}}$  élément est le minimum des valeurs **x[i]**, **y[i]**, ...  
**pmax(x,y,...)** idem pour le maximum  
**union(x,y), intersect(x,y), setdiff(x,y)**  
**union** et **intersection** d'ensembles;  
**setdiff(x,y)** trouve les éléments de **x** qui ne sont pas dans **y**  
**abs(x)** valeur absolue  
**filter(x, filter)** applique un filtre linéaire à une série temporelle; e.g., pour une moyenne mobile sur trois périodes: **filter(x, c(1/3, 1/3, 1/3))**

**na.rm=FALSE** De nombreuses fonctions mathématiques ont un paramètre **na.rm=TRUE** (non available removed) pour enlever les données manquantes (NA) avant le calcul

### Matrices

**t(x)** transposée  
**diag(x)** diagonale  
**%%** multiplication de matrices  
**solve(a,b)** trouve **x** tel que **a %% x = b**  
**solve(a)** matrice inverse de **a**  
**rowsum(x)** somme par ligne d'une matrice ou d'un objet similaire  
**colsum(x)** somme par colonne  
**rowMeans(x)** moyenne des lignes d'une matrice  
**colMeans(x)** idem pour les colonnes

### Traitement avancé des données

**apply(X,MARGIN,FUN=, ...)** applique une fonction **FUN** aux marges de **X** (**MARGIN=1** pour les lignes, **MARGIN=2** pour les colonnes); les paramètres ... sont passés à la fonction **FUN**.  
**lapply(X,FUN)** applique une fonction **FUN** à chaque élément de **X**  
**merge(x,y)** fusionne 2 data frames en utilisant leurs noms de colonnes en commun (ou en les désignant avec **by.x** et **by.y**)  
**aggregate(x,by,FUN)** divise le data frame **x** en groupes, calcule la fonction **FUN** pour chacun; **by** est une liste d'éléments de regroupement, chaque élément aussi long que le nombre de ligne de **x**  
**stack(x, ...)** transforme un tableau en plusieurs colonnes en tableau à 1 colonne, en indiquant d'où vient chaque valeur; e.g.:  
**stack(data.frame(a=1:3,b=4:6))**  
**unstack(x, ...)** inverse de **stack()**  
**reshape(x, ...)** fonction avancée (et compliquée) réorganisant en largeur ou en longueur une data.frame (e.g.: un tableau de 2 variables avec 3 années pour 4 pays contient 24 données, organisées en 2x12 ou 6x4 8x3; **reshape** convertit entre ces formats)

### Chaînes de caractères

**paste(...)** concatène des vecteurs après conversion en caractères ; **sep=** les sépare (par défaut: espace)  
**substr(x,start,stop)** extrait une sous-chaîne de caractères  
**grep(pattern,x)** renvoie les indices des éléments de **x** dans lesquels on trouve le patron **pattern**, e.g.: **grep("b", c("ab", "cd", "bz"))**  
**tolower(x)** met en minuscules  
**toupper(x)** met en majuscules  
**match(x,table)** pour chaque élément de **x**, renvoie **NA** si l'élément n'est pas trouvé dans **table**, sinon renvoie la position où il se trouve dans **table**  
**x %in% table** pour chaque élément de **x**, renvoie **TRUE** si l'élément est trouvé dans **table**, sinon renvoie **FALSE**  
**nchar(x)** nombre de caractères

### Dates et heures

La classe **Date** enregistre des dates. **POSIXct** enregistre date, heure et fuseau horaire. Les comparaisons (**>**, **<** ...), **seq()** ence, et écart de temps (**difftime()**) sont utiles. On peut enlever ou ajouter des jours à un objet **Date** (**+**, **-**).  
**as.Date(x)** convertit une chaîne de caractères en date; **as.Date("2009-12-31")+1** renvoie le 1er janvier 2010.  
**format(x)** l'inverse; on peut choisir la représentation voulue (cf. **help(strftime)**)

### Périphériques graphiques

**windows()** ouvre une fenêtre graphique sous Windows  
**x11()** idem sous GNU/linux ou MacOSX  
**pdf(file), png(file), jpeg(file), bmp(file), tiff(file)** se prépare à écrire les instructions graphiques qui suivront dans le fichier file, au format désigné (pdf ou png recommandés); **width=** et **height=** fixent les dimensions  
**dev.off()** ferme la fenêtre graphique ou le fichier graphique spécifié (par défaut: celui en cours); cf. aussi **dev.cur**, **dev.set**

### Graphiques

**plot(x)** graphique de **x** (fonction générique ayant des effets différents selon l'objet)  
**plot(x, y)** nuage de points  
**hist(x)** histogramme des fréquences de **x**  
**barplot(x)** diagramme en barres  
**pie(x)** diagramme circulaire (« camembert »)  
**boxplot(x)** diagramme en boîte [boîte à moustaches]; la boîte et son milieu montrent les 3 quartiles; les moustaches (**whisker**) un intervalle de confiance de 95% pour la médiane (s'il y a des valeurs en dehors, elles sont affichées)  
**sunflowerplot(x, y)** comme **plot(x,y)** mais les points qui se superposent exactement sont représentés avec des « fleurs » (un pétale par valeur répétée)  
**stripchart(x, method="stack")** superpose les valeurs identiques du vecteur **x**; e.g.  
**stripchart(round(rnorm(30,sd=5)), method="stack")**  
**coplot(y~x | a)** nuage des points de coordonnées **x**, **y** pour chaque valeur ou intervalle de valeur de **a**  
**mosaicplot(table(x,y))** version graphique de la table de contingence (les surfaces des carrés sont proportionnelles aux effectifs)  
**image(table(x,y))** similaire mais les effectifs influencent la couleur et non la surface  
**pairs(x)** tableau des nuages de points entre toutes les paires de colonnes de **x**  
**plot.ts(x)** pour une ou des série(s) temporelle(s) (classe **"ts"**), valeurs de **x** en fonction du temps  
**ts.plot(x)** idem mais les séries peuvent ne pas commencer ou finir en même temps  
**qqnorm(x)** nuage des quantiles observés contre quantiles théoriques; si **x** suit une loi normale, une droite; comparer **qqnorm(rnorm(100))** et **qqnorm(1:100)**  
**qqplot(x, y)** quantiles de **y** en fonction des quantiles de **x**

Les paramètres suivants sont communs à de nombreuses fonctions graphiques :

**add=TRUE** ajoute sur le graphique précédent  
**axes=FALSE** ne trace pas les axes  
**type="p"** type de représentation des coordonnées; **"p"**: points, **"l"**: lignes, **"b"**: (both) points et lignes, **"o"**: idem mais lignes sur (*over*) les points, **"h"**: bâtons, **"s"**: escaliers (données en haut des barres verticales), **"S"**: idem (données en bas des barres), **"n"**: définit la zone de coordonnées mais ne trace rien (utiliser après les commandes graphiques de bas niveau qui suivent)  
**xlim=, ylim=** limites des zones du graphique, e.g. **xlim=c(1,5)**  
**xlab=, ylab=** titre des axes (caractères)  
**main=** titre du graphique (caractères)  
**sub=** sous-titre du graphique (caractères)

## Commandes graphiques de bas niveau

Permettent de compléter un graphique existant

(éventuellement vide avec `plot(..., type="n")`)

`points(x, y)` ajoute des points (`type=` peut être utilisé)

`lines(x, y)` ajoute des lignes

`text(x, y, labels, ...)` ajoute du texte (`labels`) aux coordonnées; e.g.: `plot(x, y, type="n"); text(x, y, names)`

`segments(x0, y0, x1, y1)` trace des segments de (`x0,y0`) à (`x1,y1`)

`abline(a,b)` trace une droite (de forme  $y=a+b*x$ )

`abline(lm.obj)` trace la droite de régression du modèle linéaire `lm.obj`

`legend(x, y, legend)` ajoute une légende au point (`x,y`) avec les symboles donnés par `legend`

`axis(side)` ajoute un axe en bas (`side=1`), à gauche (`2`), en haut (`3`) ou à droite (`4`); optionnels: `at=` pour les coordonnées des graduation, `labels=` pour leur texte

`box()` encadre le graphique

`rug(x)` ajoute près de l'axe des abscisses une petite barre pour chaque valeur de `x`

`locator(n)` renvoie les coordonnées des clics de la souris après `n` clics sur le graphique

## Paramètres graphiques

`par(...)` définit les paramètres suivants pour les graphiques à venir, e.g. `par(cex=2)`; nombre de ces paramètres peuvent aussi être utilisés directement avec une commande graphique de haut ou bas niveau, e.g. `plot(x, cex=2)`; liste complète avec `help(par)`

`cex` taille du texte et des symboles par rapport à la valeur par défaut (`character expansion`)

`col` couleur(s) des symboles et lignes; e.g. `col="red"`, `"blue"` cf. `colors()`; e.g. pour créer des vecteurs de 5 couleurs, faire suivre `col=` de `gray(0:5/5)`, `rainbow(5)` ou `terrain.colors(5)`

`lty` type de ligne; **1**: pleine, **2**: tirets, **3**: pointillés, **4**: tirets-points, **5**: longs tirets, **6**: tiret-court/tiret-long; (configurable)

`lwd` largeur des lignes

`pch` type de symboles pour les points (code entier de 1 à 25, ou caractère entre ""')

`xaxt="n"` ne trace pas l'axe des abscisses

`yaxt="n"` ne trace pas l'axe des ordonnées

## Groupe de graphiques conditionnels

Pour accéder à ces fonctions, il faut faire avant:

`library(lattice)`

La formule `y~x` trace `y` en fonction de `x`. On peut faire un graphique `y~x` par sous groupe de données en indiquant l'appartenance à tel ou tel groupe par le vecteur `g1: y~x | g1`; pour toutes les combinaisons des séries de groupes `g1` et `g2: y~x | g1*g2`

`xyplot(y~x)` nuages de points

`barchart(y~x)` diagrammes en barre

`histogram(~x)` histogrammes

`bwplot(y~x)` boîtes à moustache

`stripplot(y~x)` graphique à une dimension, `x` doit être un nombre, `y` peut être un facteur

## Modèles et analyses statistiques

`lm(formula)` estimation d'un modèle linéaire;  
`formula=y~a+b` estime le modèle  $y=ax+by+c$  (mettre -1 dans la formule pour enlever la constante c);  
`summary(lm(...))` donne des informations utiles  
`glm(formula,family=)` estime un modèle linéaire généralisé; e.g. `family= binomial(link = "logit")` pour un modèle logit (cf. `?family`)

Après la formule, on peut en général préciser le nom du data.frame (`data=`) et le sous-ensemble de données (`subset=` suivi d'un vecteur de valeurs logiques)

`predict(fit,...)` fait une prédiction à partir du modèle estimé `fit` et de nouvelles données

`coef(fit)` coefficients du modèle estimé

`residuals(fit)` résidus du modèle

`fitted(fit)` valeurs prédites par le modèle

`rnorm(n, mean=0, sd=1)` distribution gaussienne (normale)

`rt(n, df)` distribution de Student (t)

`rf(n, df1, df2)` distribution de Fisher-Snedecor (F)

Ces fonctions de distribution peuvent être modifiées en changeant la première lettre pour avoir: `r` (random) pour tirer des nombres au hasard; `d`: densité de probabilité; `p`: idem cumulée; `q`: la valeur du quantile (avec le paramètre `p`:  $0 < p < 1$ )

## Programmation

Fonctions permettant d'enchaîner des opérations de manière structurée. Pour avoir de l'aide sur ces fonctions, saisir leur nom entre guillemets; e.g. `help("if")`

`function( arglist ) {expr}` définition de fonction;  
`arglist` est une liste d'arguments, `expr` est une expression exécutée; e.g.: `mafonction<-function( a, b ) {a+2*b}; mafonction(1,2) #renvoie 5`

`return(value)` mis dans `expr` lors d'une définition de fonction, indique que la fonction doit renvoyer ce résultat (si `return` est absent, la fonction renvoie la dernière valeur calculée dans `expr`)

`if(cond) {expr}` si `cond` est vrai (TRUE), évaluer `expr`  
`== != < > <= >=` opérateurs de comparaison, dans l'ordre: égal, différent, inférieur, supérieur, inférieur ou égal, supérieur ou égal; e.g. `1==1` vaut TRUE ou T; `1!=1` vaut FALSE ou F; dans les opérations avec des nombres, T est converti en 1 et F en 0 (`T-1==0` est vrai)

`if(cond) {cons.expr} else {alt.expr}` si `cond` est vrai évaluer `cons.expr` sinon évaluer `alt.expr`

`for(var in seq) {expr}` exécute l'expression pour chaque valeur de `var` prises dans une `sequence`

`while(cond) {expr}` exécute l'expression tant que la `condition` est vraie

`repeat {expr}` répète `expr` en boucle; penser à l'arrêter avec `if(...) {break}` (ou avec les touches Ctrl+C)

`break` arrête une boucle `for`, `while` ou `repeat`

`next` arrête l'itération en cours et reprend la boucle (dans le cas de `for`, avec la valeur suivante de la `sequence`)

`ifelse(test, yes, no)` pour chaque ligne/cellule de `test`, renvoie la valeur `yes` si le test est TRUE, `no` s'il est FALSE, NA s'il est indéterminé