Séance 2

Introduction au langage R

BIO109 - Dominique Gravel

Laboratoire d'écologie intégrative | Integrative Ecology Lab [IE]

Solutions de l'exercice de la séance 1

Ordonner un vecteur `X` composé de 5 lettres de Scrabble

Solution 1: Bubble sorting

```
WHILE is.sort = FALSE
  is.sort = TRUE
  FOR pos IN 1:4
    IF X[pos] > X[pos+1]
      INVERSE
      is.sort = FALSE
    END IF
  END FOR
END WHILE
```

Solution 2: Stupid ranking

```
WHILE is.sort = FALSE
  is.sort = TRUE
  MELANGE X
  FOR pos IN 1:4
    IF X[pos] > X[pos+1]
      is.sort = FALSE
    END IF
  END FOR
END WHILE
```

Rappel : les règles du `pseudo-code`

À garder en mémoire

1. N'écrivez qu'une seule instruction par ligne de pseudo-code.
2. Écrivez en lettres majuscules le verbe de chaque opération principale (la fonction utilisée)
3. Soyez explicite en nommant les opérations et les variables.
4. Soyez le plus détaillé possible (c.à.d les plus petites étapes possibles)
5. Utilisez des structures de langages de programmation connues (c.à.d `FOR`, `IF`, etc.)
6. Délimitez les étapes en formant des blocs d'instructions par l'utilisation de l'indentation.

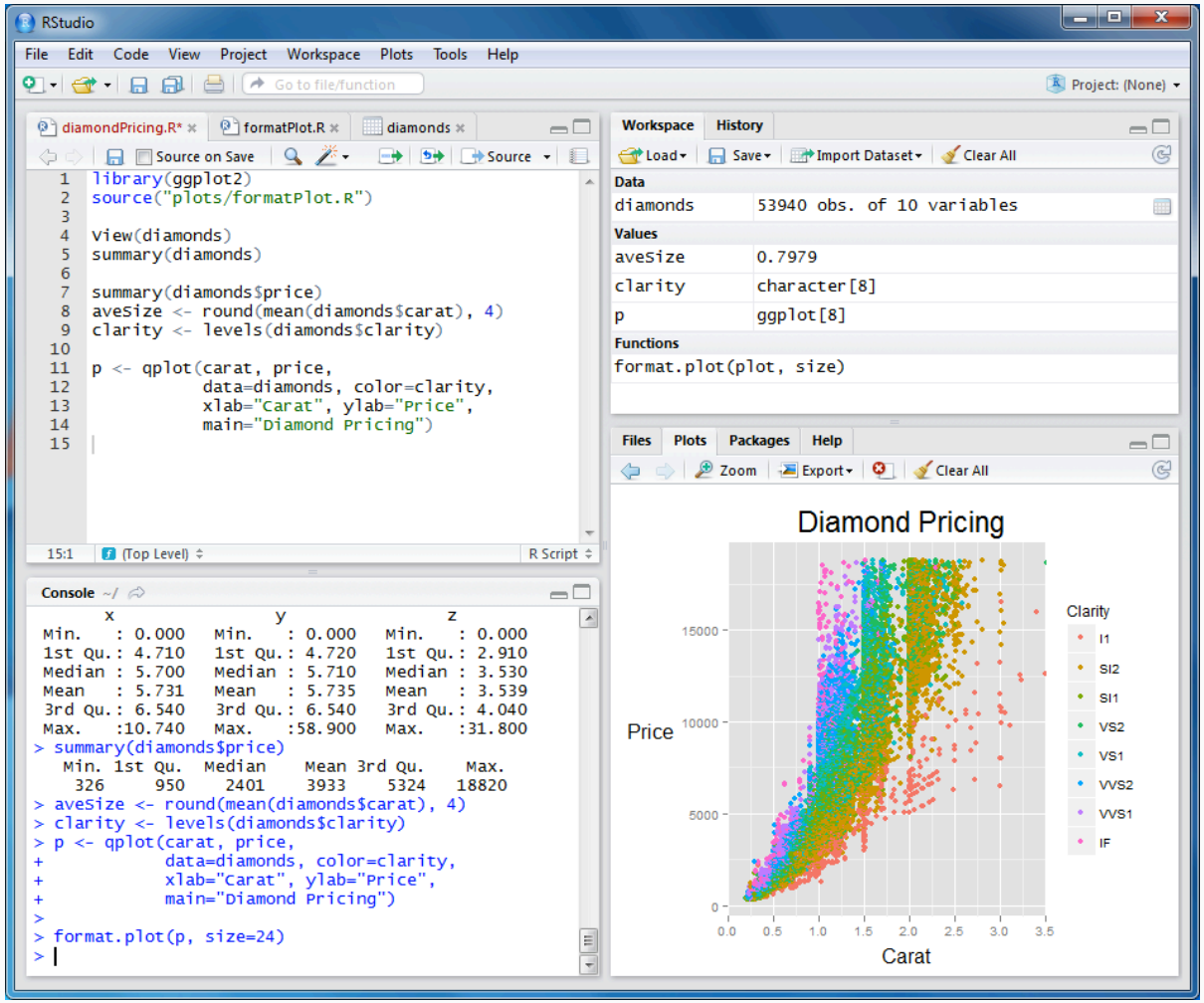
Ces règles sont générales, peu importe le langage de programmation utilisé.

Contenu du cours

1. Interagir avec R
2. Lire et écrire des objets
3. Manipuler des objets

Débuter avec R et RStudio

Utiliser R Studio



Exercice : lancer R Studio

Exercice : interaction avec la console

Vous pouvez obtenir une sortie de R en tapant des calculs dans la console.

Que donne l'opération suivante ?

```
2 + 16 * 24 - 56
```

Répertoire de travail

Savoir où on est

```
getwd()
```

```
## [1] "/home/runner/work/BIO109/BIO109/cours2"
```

Répertoire de travail

Changer le répertoire de travail

- Pour les utilisateurs Mac et Linux:

```
setwd("/home/DominiqueGravel/BIO109/cours2/donnees/")
```

- Pour les utilisateurs Windows:

```
setwd("c:/DominiqueGravel/BIO109/cours2/donnees/")
```

Le `c:` correspond à la partition du disque.

Répertoire de travail

Changer le répertoire de travail

> RStudio -> session -> changer répertoire

Environnement de travail

Obtenir le contenu du dossier dans le répertoire de travail

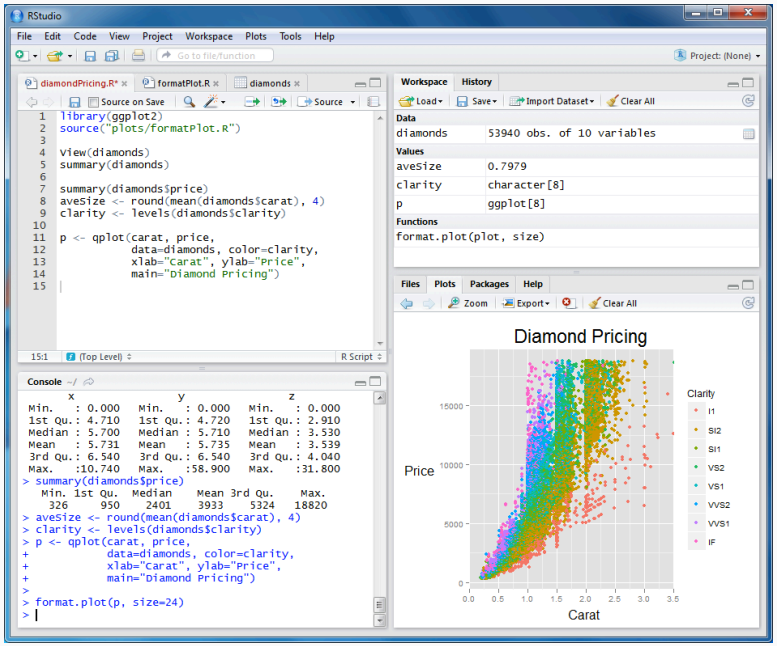
```
dir()
```

```
## [1] "assets"    "donnees"   "index.Rmd"
```

Obtenir la liste des objets en mémoire

```
ls()
```

```
## character(0)
```



Le script

C'est bien d'utiliser R comme une claculatrice glorifiée, mais on peut faire beaucoup plus !

Le script

Un script est un fichier `.R` contenant une série d'instructions et de commentaires pour réaliser des opérations sur R. Le script est utilisé pour conserver l'historique des opérations et les répéter au besoin.

Il est préférable d'utiliser uniquement le script et de documenter les commandes utilisées. Ainsi la reproductibilité est assurée.

Le script : exercice

1. Sur votre ordinateur, créer un nouveau dossier nommé `cours2`
2. Dans RStudio, définir ce dossier comme le répertoire de travail

> RStudio -> session -> changer répertoire

3. Créer un nouveau script appelé `MonScript.R`

> RStudio -> fichier -> nouveau fichier -> R script

Le concept d'objet

Un objet peut contenir de l'information de toute nature: un tableau de données, les résultats d'une analyse, une figure, une fonction

La commande pour créer un objet `a` est composé de trois parties:

1. une valeur d'intérêt
2. un identifiant (soit un nom d'objet)
3. l'opérateur d'assignation

```
a ← c(1,2)
a
```

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

Le concept d'objet

- Notons que les opérateurs `←` et `=` sont équivalents pour attribuer une valeur à un objet.

La recommandation est donc de limiter le signe `=` à des opérations mathématiques et d'utiliser `←` pour l'assignation de valeurs.

Le commentaire

Un script peut contenir des commentaires

```
# Une première opération
3 + 5
```

Exercice 2

Dans le script `MonScript.R`, ajouter et puis exécuter les lignes suivantes:

```
# Une première opération
3 + 5

# Une deuxième opération
x * 10
```

Les erreurs et avertissements

Error

Arrêt de l'exécution

```
# Une première opération
3 + 5

# Une deuxième opération
x * 10
```

La sortie dans la console :

```
> # Une première opération
> 3 + 5
[1] 8
>
> # Une deuxième opération
> x * 10
Erreur : objet 'x' introuvable
```

Les erreurs et avertissements

Warning

Mise en garde sans arrêter l'exécution

```
# Un vecteur
x ← "end"

# Une opération
as.numeric(x)
```

La sortie dans la console :

```
> as.numeric(x)
[1] NA
Message d'avis :
NAs introduits lors de la conversion automatique
```

Les types de valeurs assignables à un objet

Les types de valeurs

- `numérique`
- `caractère`
- `booléen`
- `facteur`

Assigner une valeur de type `numérique`

Les données numériques peuvent être déclarées de différentes manières

```
a ← 1
a
```

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

```
b ← 1.1
b
```

```
## [1] 1.1
```

```
c ← 1e-6
c
```

```
## [1] 1e-06
```

Assigner une valeur de type `caractère`

On peut également déclarer des séries de caractères

```
test ← "test"
test
```

```
## [1] "test"
```

Important - Les valeurs de type `caractère` doivent être placées entre guillemets. Cette nomenclature permet de différencier une valeur du nom d'un objet.

Assigner une valeur de type `booléen`

Le type `booléen` (logical) permet de représenter les états `TRUE` (1) et `FALSE` (0) et de faire des opérations mathématiques sur ces objets.

```
vrai ← TRUE
faux ← FALSE
vrai
```

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

```
vrai + faux
```

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

Assigner une valeur de type `facteur`

R est d'abord un langage utilisé pour les statistiques, et par conséquent on y retrouve un type de données utilisé pour la réalisation de tests d'hypothèse, qui ne sont pas standard à tous les langages. Les facteurs sont des données de catégories comme des traitements expérimentaux.

```
MesFacteurs ← factor(c("trait1","trait2","trait3","trait2","trait3","trait3"))
MesFacteurs
```

```
## [1] trait1 trait2 trait3 trait2 trait3 trait3
## Levels: trait1 trait2 trait3
```

Conversion entre types de valeur

Conversion entre types de valeur

R permet de convertir des objets en différents types de données lorsque le contenu le permet.

Conversion vers les types `numérique` et `caractère`:

```
as.numeric(c("4","6"))
```

```
## [1] 4 6
```

```
as.character(c(4,6))
```

```
## [1] "4" "6"
```

Conversion entre types de valeur

Conversion vers les types `facteurs` et nombre `entier`:

```
as.integer(2.6)
```

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

```
as.factor(c("4","6"))
```

```
## [1] 4 6
## Levels: 4 6
```

Conversion entre types de valeur

Conversion vers le type `booléen`:

```
as.logical(0)
```

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

```
as.logical(1)
```

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

```
as.logical(2)
```

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

Les types d'objets

Les types d'objets

Depuis le début du cours, nous avons assigné une seule valeur à un objet. Lorsqu'un objet contient une seule valeur, on dit que c'est un objet `scalaire`.

```
a ← 1
a
```

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

Cependant, un objet peut cependant contenir plusieurs valeurs. Il sera défini alors comme un `vecteur`.

type d'objets: les `vecteurs`

Le vecteur est un des objets les plus importants de R.

La façon la plus simple de déclarer un vecteur est:

```
MonPremierVecteur ← c(1,2,3,4,5)
MonPremierVecteur
```

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

`c` signifie combiner

Note - D'autres méthodes seront présentées plus loin.

type d'objets: les `vecteurs`

La force de R repose dans sa capacité à réaliser des opérations `vectorielles`.

Il est ainsi possible d'additionner simplement deux vecteurs d'une même dimension de la façon suivante:

```
MonPremierVecteur ← c(1,2,3,4,5)
MonDeuxiemeVecteur ← c(11,12,13,15,16)
MonPremierVecteur + MonDeuxiemeVecteur
```

```
## [1] 12 14 16 19 21
```

Exercice

Créez un vecteur contenant les valeurs `[-1,2,5,9]`. Vous pouvez calculer la racine carrée de ces nombres au moyen de la fonction `sqrt()`

Types d'objets: les `vecteurs`

On peut accéder à une position sur le vecteur au moyen d'un `index`, indiqué par les `[]`

```
MonPremierVecteur ← c(1,2,3,4,5)
MonPremierVecteur[3]
```

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

On obtient un `NA` si on tente d'accéder à une position qui n'existe pas:

```
MonPremierVecteur ← c(1,2,3,4,5)
MonPremierVecteur[6]
```

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

Exercice

1. Au moyen du vecteur créé précédemment, calculer le produit des valeurs aux positions `2` et `4`.
2. Sélectionner les valeurs du vecteur qui sont supérieures à `3`.

Types d'objets: les `vecteurs`

On peut également accéder à plusieurs positions en même temps:

```
MonPremierVecteur[c(1,3,5)]
```

```
## [1] 1 3 5
```

Types d'objets: les `vecteurs`

On obtient la dimension d'un vecteur ainsi:

```
length(MonPremierVecteur)
```

```
## [1] 5
```

Types d'objets: les `matrices`

La `matrice` est un objet qui contient plusieurs vecteurs de même dimension. R est également optimisé pour réaliser des opérations mathématiques et de manipulation de données sur ce type d'objets.

La commande de base pour créer une matrice est `matrix()`

```
MaMatrice ← matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
MaMatrice
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Types d'objets: les `matrices`

On accède à la ligne `i` et la colonne `j` au moyen de la commande suivante

```
MaMatrice[1,2]
```

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

Types d'objets: les `matrices`

On peut aussi accéder à des lignes ou des colonnes entières

```
MaMatrice[1,]
```

```
## [1] 1 3 5
```

```
MaMatrice[,1]
```

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

Types d'objets: les `matrices`

On obtient la dimension de la matrice ainsi

```
nrow(MaMatrice)
```

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

```
ncol(MaMatrice)
```

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

Types d'objets: les `matrices`

Les noms de colonnes et de lignes peuvent être modifiés

```
colnames(MaMatrice) ← c("A","B","C")
rownames(MaMatrice) ← c("X","Y")
MaMatrice
```

```
##   A B C
## X 1 3 5
## Y 2 4 6
```

Exercice

Créez une matrice de 2 lignes et 5 colonnes remplies de chiffres tirés au hasard. Pour ce faire, vous pouvez utiliser la fonction `runif()`. Calculez la somme de la première colonne au moyen de la fonction `sum()`

Astuce : pour obtenir de l'aide sur une fonction, essayez

```
?runif()
```

Type d'objets: les `data.frame`

Le `data.frame` se décrit comme un tableau de données, avec les rangées `i` et les colonnes `j`, ainsi que des noms de colonnes.

```
df ← data.frame(a = c(1:3), b = c(11:13))
df
```

```
##   a  b
## 1 1 11
## 2 2 12
## 3 3 13
```

```
df$a
```

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

On peut indexer une colonne directement avec le symbole "$"

Type d'objets: les `data.frame`

Un `data.frame` peut avoir différents types de valeur pour chaque colonne

```
df2 ← data.frame(a = c(1:3), traitement = factor(c("trait1","trait2","trait3")))
df2
```

```
##   a traitement
## 1 1     trait1
## 2 2     trait2
## 3 3     trait3
```

Cette capacité à entreposer plusieurs types de valeur est ce qui distingue un `data.frame` d'une matrice.

Type d'objets: les `listes`

R peut organiser également des collections d'objets sous forme de liste. Ces collections peuvent être très hétérogènes et rassembler par exemple, des vecteurs et matrices, mais aussi des objets plus complexes et hiérarchiques comme des résultats d'analyses statistiques.

La création et l'indexation des listes sont très similaires à celle des vecteurs et matrices

```
MaListe ← list()
MaListe[[1]] ← c(1,2)
MaListe[[2]] ← matrix(c("A","B","C","D"), nrow = 2, ncol = 2)
```

`MaListe[[1]]` permet d'obtenir l'objet contenu par le premier niveau de la liste.

Type d'objets: les `listes`

Et de même on peut nommer les items d'une liste

```
names(MaListe) ← c("Element1","Element2")
MaListe
```

```
## $Element1
## [1] 1 2
##
## $Element2
##      [,1] [,2]
## [1,] "A"  "C"
## [2,] "B"  "D"
```

Interagir avec ses fichiers

Lire un fichier

La lecture de fichiers est souvent l'étape la plus frustrante lorsque l'on travaille avec R.

Pour ce faire, nous utiliserons les fichiers sous format `csv` (*Comma Separated values*) ou `txt` qui sont des fichiers de texte brut dont chaque ligne dispose d'un séparateur de colonne.

Lire un fichier

Si j'ouvre le fichier arbres.csv avec un bloc note, j'obtiens les 5 premières lignes suivantes:

```
id_bor;borx;bory;arbre;esp;multi;mort;dhp
0-0;0;0;34501;acpe;FAUX;FAUX;82
0-0;0;0;34502;acpe;VRAI;FAUX;26
0-0;0;0;34502;acpe;VRAI;FAUX;98
0-0;0;0;34503;acpe;FAUX;FAUX;73
```

On voit ici le point virgule comme séparateur de colonne. La première ligne correspond au nom des colonnes.

Note: Garder à l'esprit que connaitre la structure du fichier facilite grandement l'importation des données dans R.

Lire un fichier Excel

Les fichiers `.xlsx` peuvent être lus au moyen de librairies spécialisées, mais il est préférable sous R d'utiliser un format multi-plateforme ouvrable sans logiciels propriétaires.

Ouvrir un fichier `.csv` dans R

Nous allons utiliser le fichier arbres.csv pour cet exemple.

On peut lire le fichier au moyen de la commande

```
arbres ← read.table(file = "donnees/arbres.csv", header = T, dec = ".", sep = ";")
head(arbres)
```

```
##   id_bor borx bory arbre  esp multi mort dhp
## 1    0-0    0    0 34501 acpe  FAUX FAUX  82
## 2    0-0    0    0 34502 acpe  VRAI FAUX  26
## 3    0-0    0    0 34502 acpe  VRAI FAUX  98
## 4    0-0    0    0 34503 acpe  FAUX FAUX  73
## 5    0-0    0    0 34504 acpe  FAUX VRAI  28
## 6    0-0    0    0 34506 fagr  FAUX FAUX  26
```

Exercice : Ouvrir un fichier `.csv` dans R

1. Créer un dossier dans votre répertoire de travail `cours2` et nommer le `donnees`.
2. Télécharger le fichier arbres.csv et le sauver dans le dossier `donnees`.
3. Ouvrir le fichier avec `arbres ← read.table(file = "donnees/arbres.csv", header = T, dec = ".", sep = ";")`

Lire un fichier `.csv` : anatomie des arguments

- `file` : nom du fichier à lire
- `header` : indique s'il y a un entête avec les noms de colonnes
- `dec` : caractère utilisé pour délimiter les décimales
- `sep` : caractère utilisé pour séparer les colonnes
- `quote` : spécifie si les chaines de caractère sont entourées par des guillemets.

Note : l'objet retourné est par défaut un `data.frame`. Il peut être ensuite converti, au besoin.

Lire un fichier: les erreurs courantes

- Mauvais nom de fichier
- Mauvais répertoire de travail
- Guillemets
- Type de séparateur de colonnes
- Présence de points
- Nom des colonnes
- Entrées vides
- Présence de caractères invisibles
- Mauvais encodage des données
- Type de données et d'objets

Écrire des fichiers `csv` et `txt`

Il y a une grande diversité de façons d'enregistrer sur le disque des objets provenant de R. La façon la plus simple, versatile et qui permet les échanges entre différents logiciels et systèmes d'exploitation est d'écrire sous format `csv`.

La syntaxe pour écrire une `matrice` ou un `data.frame` est aussi simple que pour lire un fichier `csv`.

```
mat2 ← matrix(runif(n = 10,min = 0,max = 1),nrow = 5,ncol = 10)
write.table(mat2, file = "mat.csv")
```

Écrire des fichiers `.Rdata` avec `save()`

Parfois les objets que l'on souhaite enregistrer ont une structure plus complexe qu'un tableau de données. R permet d'enregistrer ces objets dans un format qui lui est unique, le `.Rdata`. Ces objets sont compressés pour minimiser l'espace disque et ne peuvent être lus uniquement par R.

```
MaListe ← list()
MaListe[[1]] ← 1
MaListe[[2]] ← c(1:10)
save(MaListe, file = "MaListe.Rdata")
```

NOTE - `save()` permet également de sauvegarder plusieurs objets.

```
save(MaListe,MonDataFrame, file = "MaListe.Rdata")
```

Lire des fichiers `.Rdata` avec `load()`

Puisque le fichier `.Rdata` est spécifique à R, il s'agit peut-être du format le plus facile à lire puisque R prend en charge la mise en forme de l'objet, les noms et les types de données

```
load("MaListe.Rdata")
ls()
```

```
##  [1] "a"                 "arbres"            "b"
##  [4] "c"                 "df"                "df2"
##  [7] "faux"              "MaListe"           "MaMatrice"
## [10] "MesFacteurs"       "MonDeuxiemeVecteur" "MonPremierVecteur"
## [13] "test"              "vrai"
```

Lire des fichiers: `load()` et `.Rdata`

Faites attention, si le nom de l'objet contenu dans le fichier `.Rdata` est le même qu'un objet en mémoire, il va écraser ce premier objet.

```
MaListe ← list("Hello","World")
load("MaListe.Rdata")
MaListe
```

```
## [[1]]
## [1] 1
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10

## [1] TRUE
```

Pour éviter cette erreur, il faut utiliser les fonctions `saveRDS()` et `readRDS()`.

Quelques commandes utiles

Il est important de vérifier la structure de l'objet après son importation dans l'environnement R.

```
head(arbres, n = 3)
```

```
##   id_bor borx bory arbre  esp multi mort dhp
## 1    0-0    0    0 34501 acpe  FAUX FAUX  82
## 2    0-0    0    0 34502 acpe  VRAI FAUX  26
## 3    0-0    0    0 34502 acpe  VRAI FAUX  98
```

```
tail(arbres, n = 3)
```

```
##        id_bor borx bory arbre  esp multi mort dhp
## 17633 180-980  180  980 17271 piru  FAUX FAUX 125
## 17634 180-980  180  980 17272 piru  FAUX FAUX 190
## 17635 180-980  180  980 17273 beal  FAUX FAUX 210
```

Quelques commandes utiles

Il est important de vérifier la structure de l'objet après son importation dans l'environnement R.

```
str(arbres)
```

```
## 'data.frame':    17635 obs. of  8 variables:
##  $ id_bor: Factor w/ 499 levels "0-0","0-100",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ borx  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ bory  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arbre : int  34501 34502 34502 34503 34504 34506 34507 34509 34510 34511 ...
##  $ esp   : Factor w/ 7 levels "abba","acpe",..: 2 2 2 2 2 6 2 2 2 2 ...
##  $ multi : Factor w/ 2 levels "FAUX","VRAI": 1 2 2 1 1 1 1 1 1 1 ...
##  $ mort  : Factor w/ 2 levels "FAUX","VRAI": 1 1 1 1 2 1 1 1 1 1 ...
##  $ dhp   : int  82 26 98 73 28 26 29 18 24 34 ...
```

Quelques commandes utiles

`summary()` permet d'obtenir un résumé du contenu de chaque colonne

```
summary(arbres)
```

```
##      id_bor           borx             bory           arbre          esp
##  80-80  :  227   Min.   :  0.00   Min.   :  0.0   Min.   :    1   abba:2596
##  100-20 :  224   1st Qu.: 60.00   1st Qu.:160.0   1st Qu.: 4358   acpe:1864
##  100-60 :  204   Median : 80.00   Median :400.0   Median : 8777   acsa:3326
##  80-260 :  196   Mean   : 88.99   Mean   :437.9   Mean   : 9074   beal:3995
##  80-240 :  189   3rd Qu.:120.00   3rd Qu.:700.0   3rd Qu.:13186   bepa:2080
##  80-20  :  184   Max.   :180.00   Max.   :980.0   Max.   :64508   fagr:2785
##  (Other):16411                                                    piru: 989
##   multi          mort             dhp
##  FAUX:16920   FAUX:15564   Min.   :   0.0
##  VRAI:  715   VRAI: 2071   1st Qu.: 105.0
##                            Median : 165.0
##                            Mean   : 183.7
##                            3rd Qu.: 254.0
##                            Max.   :3015.0
##
```

Quelques commandes utiles

Très souvent, on souhaite réaliser un sommaire de nos données. La fonction `summary()` est fort utile, mais parfois on souhaite avoir d'autres informations que la moyenne (p. ex. la variance). Dans ce cas, la fonction `table()` est recommandée.

```
n_sp ← table(arbres$esp)
n_sp
```

```
##
## abba acpe acsa beal bepa fagr piru
## 2596 1864 3326 3995 2080 2785  989
```

Exercice de manipulation des données

- Télécharger le fichier quadrats.csv
- Lire le fichier au moyen de R
- Vérifier que le fichier a le format approprié
- Créer un vecteur `AbondanceMoyenne` d'une longueur correspondant au nombre d'espèces dans les données quadras
- Calculer l'abondance moyenne de chaque espèce au moyen de la fonction 'mean' et l'inscrire dans chaque position de `AbondanceMoyenne`

Note: vous pouvez explorer les données avec `summary` et pour les avancés. Les plus avancés peuvent explorer la commande `apply`...

Ressources

Ressources dans R

`?` pour obtenir de l'aide sur une fonction

Ressources en ligne :

- stackoverflow.com
- R for Data Science

Manipulation des données

Différents outils pour générer des données

```
seq(from = 1, to = 10, by = 0.5)
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0
## [16]  8.5  9.0  9.5 10.0
```

```
rep(c(1:3), times = 3)
```

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

Combiner des objets

```
v1 ← c(1:3)
v2 ← c("A", "B", "C")
rbind(v1,v2)
```

```
##    [,1] [,2] [,3]
## v1 "1"  "2"  "3"
## v2 "A"  "B"  "C"
```

```
cbind(v1,v2)
```

```
##      v1  v2
## [1,] "1" "A"
## [2,] "2" "B"
## [3,] "3" "C"
```

Fonctions utiles: trier des objets

```
tri ← runif(n = 10, min = 0, max = 100)
tri
```

```
##  [1] 58.120132  6.013674 82.931924 22.438037 80.061905 37.222292 92.410323
##  [8] 75.411327  7.681238 47.109242
```

```
sort(tri)
```

```
##  [1]  6.013674  7.681238 22.438037 37.222292 47.109242 58.120132 75.411327
##  [8] 80.061905 82.931924 92.410323
```

Fonctions utiles: obtenir des rangs

```
rang ← runif(n = 10, min = 0, max = 100)
rang
```

```
##  [1] 37.23250 38.96754 41.53340 31.20221 74.87029 24.31852 44.16041 62.08126
##  [9] 96.29489 40.05407
```

```
rank(rang)
```

```
##  [1]  3  4  6  2  9  1  7  8 10  5
```

Fonctions utiles: valeurs uniques

```
uq ← c(1,2,5,7,4,3,2,1,10,5,8)
uq
```

```
##  [1]  1  2  5  7  4  3  2  1 10  5  8
```

```
unique(uq)
```

```
## [1]  1  2  5  7  4  3 10  8
```

Sous-échantillonner des objets

Parfois, on souhaite avoir seulement une partie des données contenues dans un objet. La fonction `subset()` est fort pratique pour réaliser cette opération.

Ici par exemple, dans l'exemple de Sutton, on souhaite étudier seulement les quadrats possédant des érables à sucre de plus de 200 mm:

```
arbres ← read.table(file="donnees/arbres.csv", header=TRUE, stringsAsFactors=FALSE,
sub_quadrats ← subset(arbres, arbres$esp == "acsa" & arbres$dhp ⩾ 200)
head(sub_quadrats, n=2)
```

```
##    id_bor borx bory arbre  esp multi mort dhp
## 14    0-0    0    0 34516 acsa  FAUX FAUX 371
## 22    0-0    0    0 34524 acsa  FAUX FAUX 388
```

Le "==" est utilisé pour sélectionner les individus qui sont des acsa et le symbole "&" pour combiner deux conditions.

Fonction source

Il est possible d'exécuter un script sans avoir à l'ouvrir.

```r
source("MonScript.R")
```

Exercice de fin de séance

Le fichier quadrats.csv est un sommaire de données individuelles, où la présence de chaque espèce est mesurée. Ces données se trouvent dans arbres.csv. Dans le cadre de ce projet, on s'intéresse à la distribution de l'érable à sucre et des autres espèces tout au long du gradient d'élévation de la parcelle. Pour cet exercice, on vous demande de:

1. Charger les données "arbres"
2. Délimiter cinq zones au sein du gradient d'élévation : 0-200m, 201-400m, 401-600m, 601-800m, 801-1000m
3. Pour chacune de ces zones, calculer le nombre de tiges de chaque espèce
4. Enregistrer les résultats dans un tableau avec 5 rangées et S colonnes

On vous demande de rédiger un script qui réalisera l'ensemble de ces étapes, de la lecture des données à l'enregistrement du tableau final.