



Atelier 1: Introduction à R

Série d'ateliers R du CSBQ

Centre de la Science de la Biodiversité du Québec



À propos de cet atelier



Objectifs d'apprentissage

1. Comprendre ce que R et RStudio sont
2. Utiliser R comme une calculatrice
3. Manipuler les objets dans R
4. Installer et utiliser les packages R

1. Comprendre ce que R et RStudio sont

Introduction

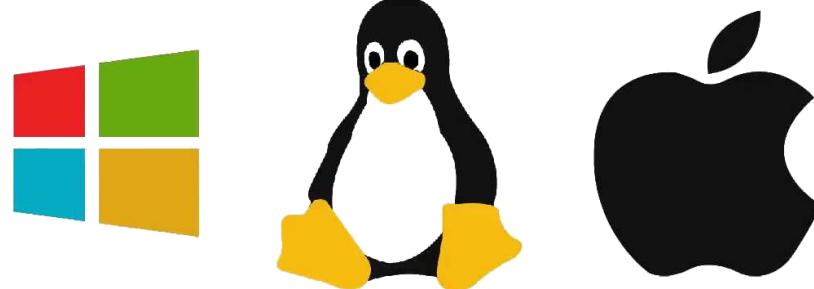
Qu'est-ce que R?

R est un langage de programmation open source conçu pour l'analyse statistique, l'exploration de données et la visualisation de données.



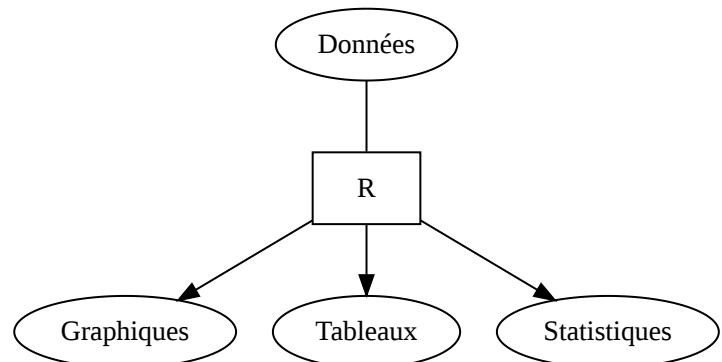
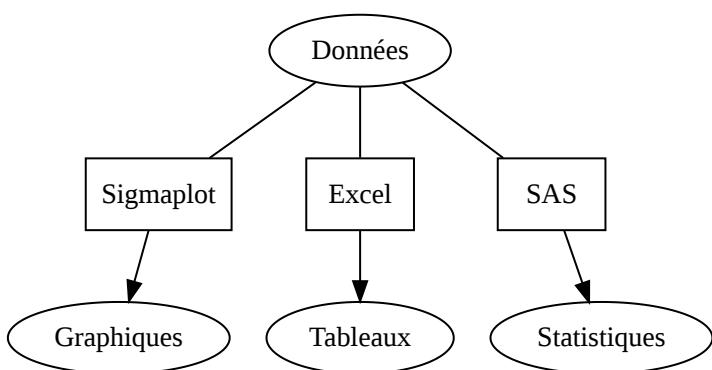
Pourquoi utiliser R?

- **R est en source libre, open source**
 - Amélioré **par** la communauté d'utilisateur, **pour** la communauté!
- **Gratuit!**
- **Compatible!**
 - R fonctionne avec la majorité des systèmes d'exploitation



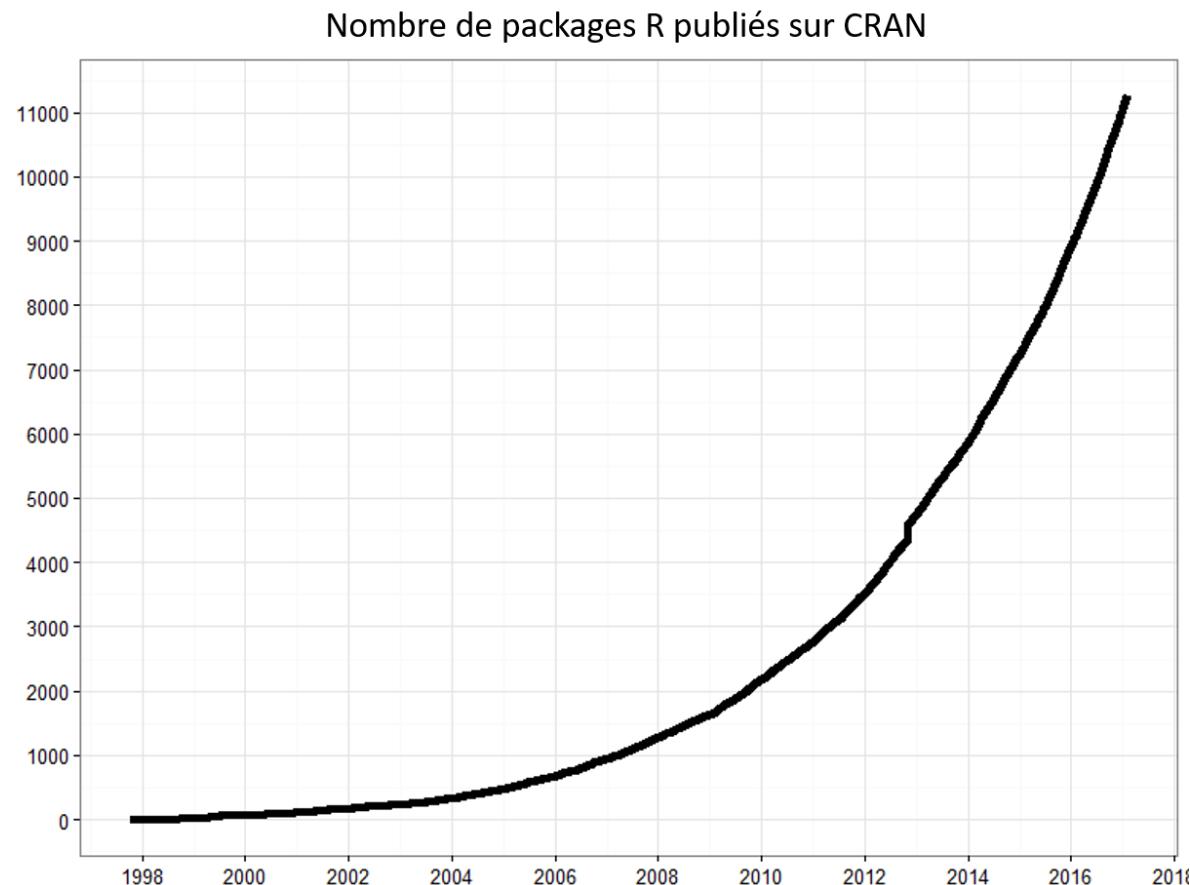
Pourquoi utiliser R?

- Ce que les gens faisaient traditionnellement pour analyser leurs données:
- R permet de tout (ou presque) faire avec un seul programme!



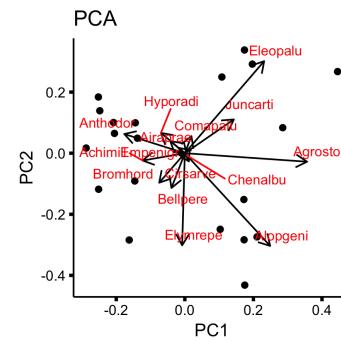
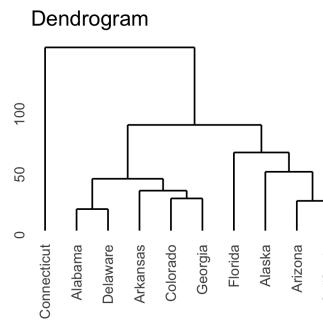
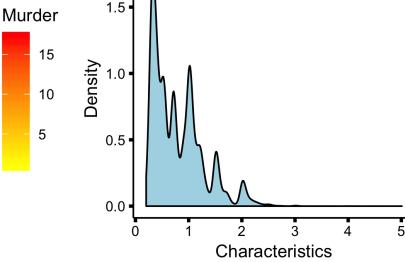
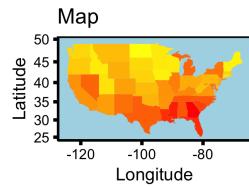
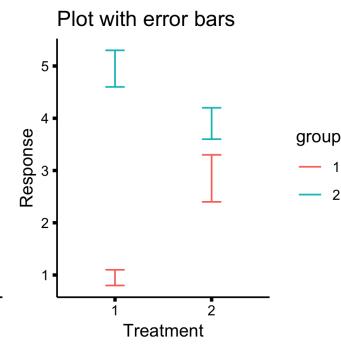
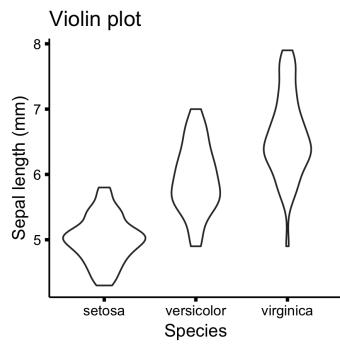
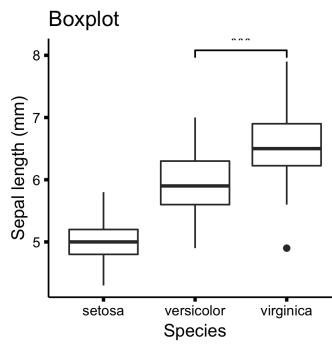
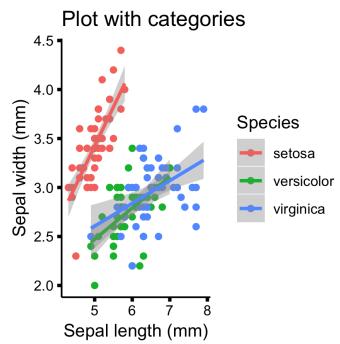
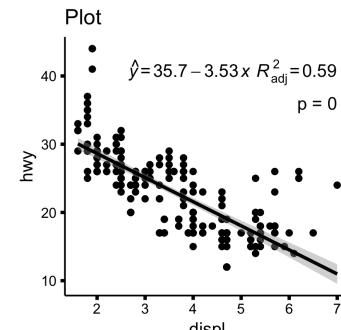
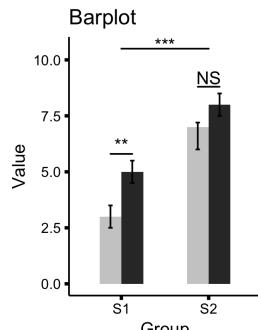
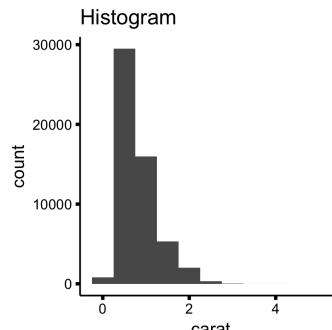
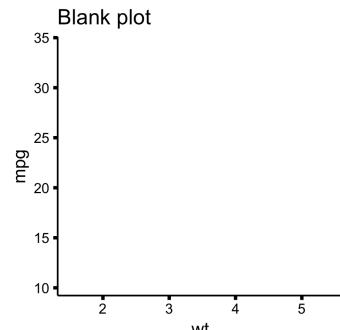
Pourquoi utiliser R?

- De plus en plus de scientifiques l'utilisent chaque année!
- Des possibilités qui augmentent chaque année



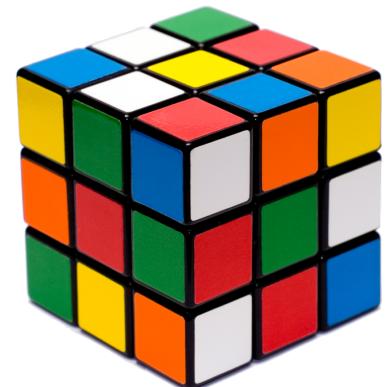
Visualisation de données!

Tous ces graphiques ont été faits avec R!



Défi

- Tout au long de ces ateliers, vous aurez à compléter plusieurs défis qui sont indiqués par des Rubik's Cube
- Pendant les défis, collaborez avec vos voisins!



Défi 1

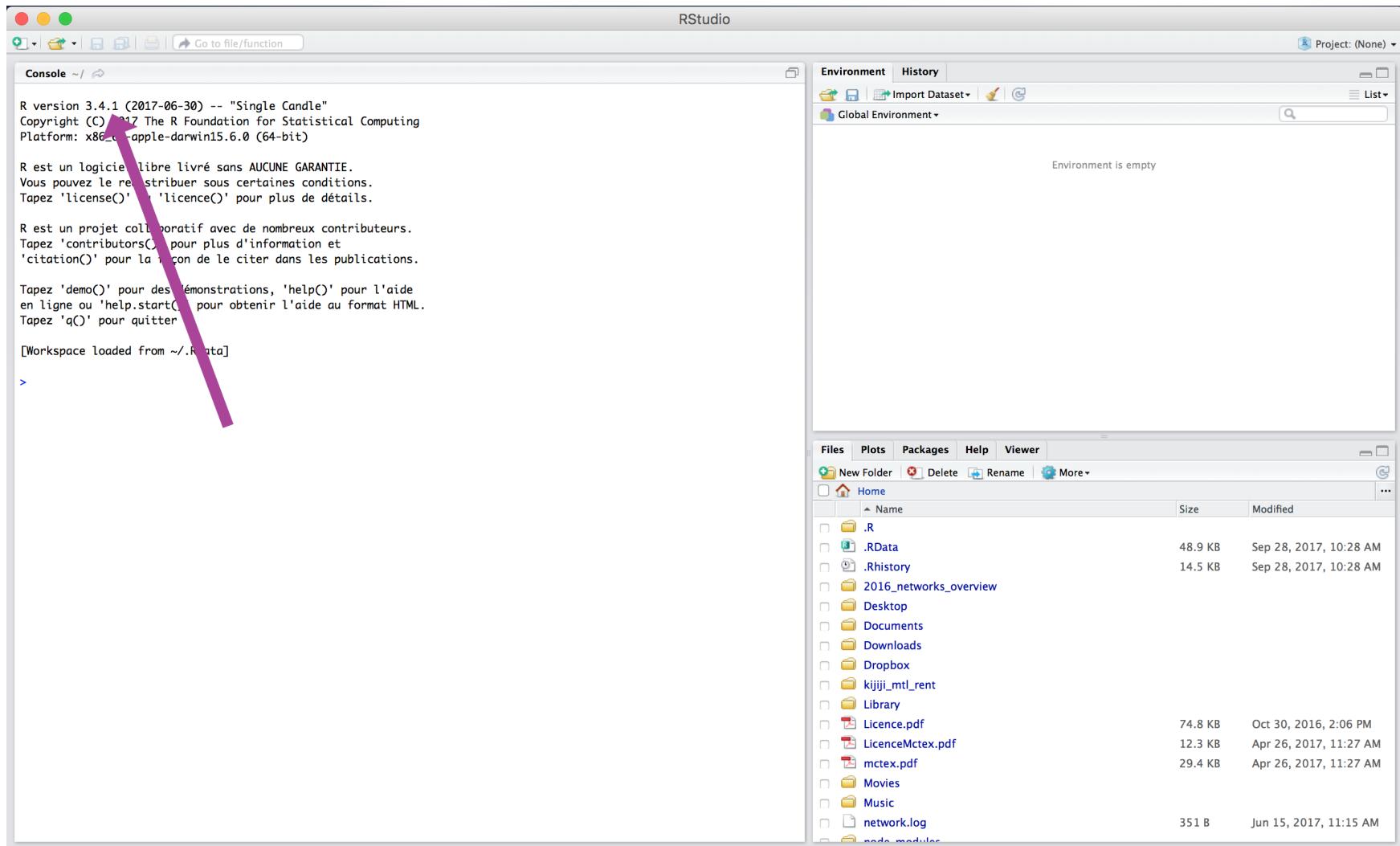


Ouvrez Rstudio



RStudio®

La console Rstudio



Note pour les utilisateurs Windows

Si la restriction `unable to write on disk` apparaît quand vous ouvrez R Studio ou que vous installez une bibliothèque,

Pas de panique!

On a la solution!

- Fermez l'application.
- Faites un clic-droit sur l'icône RStudio et choisissez: "Execute as administrator" pour ouvrir le programme.

Comment lire la console?

Le texte dans la console ressemble typiquement à ceci:

Vous devez toujours appuyer sur "enter" pour que le code soit exécuté dans la console.

```
sortie  
# [1] "Ceci est la sortie"
```

Comment lire la console?

sortie

```
# [1] "Ceci est la sortie"
```

Que signifient les crochets `[]` dans la sortie?

Comment lire la console?

sortie

```
# [1] "Ceci est la sortie"
```

Les crochets vous aident à localiser où vous êtes dans la sortie

```
# [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
# [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Erreur et avertissement

Warning message

Met en garde l'utilisateur sans arrêter l'exécution d'une fonction.

Bien que la fonction puisse vous donner une réponse, il pourrait y avoir un problème avec vos entrées. Ainsi, le calcul pourrait être erroné.

Error message

Arrête l'exécution en cours car R est incapable de réaliser le calcul.

Il indique un problème dans votre code.

Pour résoudre une erreur, les internets sont votre meilleur ami!

2. Utiliser R comme une calculatrice

Opérations de base

Opérateurs arithmétiques

- Additions et soustractions

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

```
10 - 1  
# [1] 9
```

- Multiplications et divisions

```
2 * 2  
# [1] 4
```

```
8 / 2  
# [1] 4
```

- Exposants

```
2^3  
# [1] 8
```



Défi 2

Utilisez R pour effectuer l'opération suivante:

$$2 + 16 * 24 - 56$$

Astuce: Le symbole `*` est utilisé pour multiplier



Défi 2: Solution

$$2 + 16 * 24 - 56$$

```
2 + 16 * 24 - 56  
# [1] 330
```



Défi 3

Utilisez R pour effectuer l'opération suivante:

$$2 + 16 * 24 - 56 / (2 + 1) - 457$$

Astuce: Pensez à l'ordre des opérations



Défi 3: Solution

Utilisez R pour effectuer l'opération suivante:

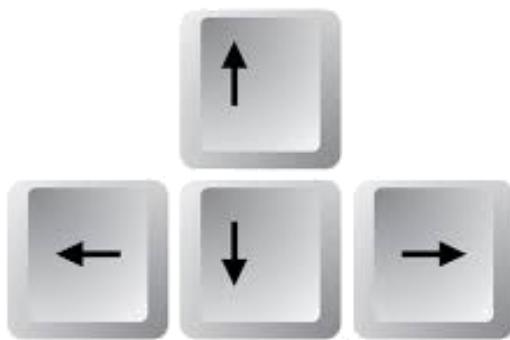
$$2 + 16 * 24 - 56 / (2 + 1) - 457$$

```
2 + 16 * 24 - 56 / (2 + 1) - 457  
# [1] -89.66667
```

Notez que R respecte l'ordre des opérations

Astuce sur la ligne de commande R

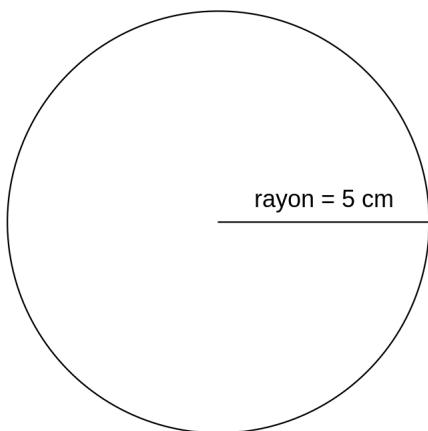
Utilisez les touches fléchées \uparrow et \downarrow pour copier et reproduire les commandes précédentes.





Défi 4

Quel est l'aire de ce cercle dont le rayon est de **5 cm**?





Défi 4: Solution

Quel est l'aire de ce cercle dont le rayon est de **5 cm**?

```
3.1416 * 5^2  
# [1] 78.54
```

Astuce: Notez que **R** a des constantes intégrées, telle que π , vous pouvez donc écrire:

```
pi * 5^2  
# [1] 78.53982
```

3. Manipuler les objets dans R

Objet

Objet

- Un concept très important en R!
- Vous pouvez stocker des valeurs dans un objet nommé à l'aide de l'opérateur d'assignation <-

nom_objet <- valeurs

- L'opérateur <- représente une flèche vers l'endroit où stocker l'information.
- Les valeurs à **droite** (valeurs) sont assignées à l'objet à **gauche** (nom_objet) grâce à l'opérateur d'assignement <-.
- Faites attention de ne pas laisser d'espace entre le symbole < et -.

Bien qu'il soit aussi possible d'utiliser le signe =, l'opérateur <- est généralement la méthode standard pour l'assignation d'information à un objet.

Nom d'objet

- Les noms d'objets peuvent inclure :

Type	Symbol
Lettres	a-z A-Z
Nombres	0-9
Point	.
Tiret bas	_

- Les noms d'objets doivent **toujours commencer par une lettre**.
- **R** est **sensible à la casse**, les noms `Data1` et `data1` ne sont pas équivalents.
- Les **caractères spéciaux sont interdits!** (@, /, etc.)

Bonnes pratiques du code R

Nom:

- Essayez d'avoir des noms courts et explicites pour vos variables. Nommer une variable `var` n'est pas très instructif.
- Utilisez un trait de soulignement `_` pour séparer les mots d'un nom ou essayez d'être constant!
- Évitez les noms d'objets de fonction qui existe dans R (e.g. `c` ou `table`)

Espace:

- Ajoutez des espaces autour de tous les opérateurs (`=`, `+`, `-`, `<-`, etc.) pour rendre le code plus lisible
- Mettez un espace après une virgule, mais jamais avant (comme en français).

```
moy_x <- (2 + 6) / 2
```

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



Défi 5

Créez un objet avec une valeur de $1+1.718282$ (**e ou le nombre d'Euler**) et nommez le `valeur_euler`.

Défi 5: Solution



Créez un objet avec une valeur de $1+1.718282$ (le nombre d'Euler) et nommez le `valeur_euler`.

```
valeur_euler <- 1 + 1.718282
```

```
valeur_euler  
# [1] 2.718282
```



Défi 6

Créez un deuxième objet avec un nom qui commence par un nombre. Qu'est-ce qui se produit?



Défi 6: Solution

Créez un deuxième objet avec un nom qui commence par un nombre. Qu'est-ce qui se produit?

La création d'un nom d'objet commençant par un nombre renvoie l'erreur suivante:

```
Error: unexpected symbol in "[nom de votre objet]"
```

Astuce sur la ligne de commande R

- Utilisez la touche de tabulation pour effectuer une saisie semi-automatique
- Cela permet d'éviter les fautes d'orthographe et accélère la saisie des commandes

Essayons!

Astuce sur la ligne de commande R

- Tapez `val`
- Appuyez sur tab
- utilisez les flèches et la touche enter pour sélectionner la bonne saisie semi-automatique

3. Manipulate objects in R

Types de structures de données en R

Types de structures de données en R

Types de structures de données en R

vecteur

- Combinaison d'éléments du même type de données.
- Structure de base; tous les objets sont formés d'un ou plusieurs vecteurs.

matrice

- Objet 2D généré en combinant plusieurs vecteurs du même type.
- On peut faire des opérations sur des matrices en R (+, -, *, ...).

array

- Objet 3D généré en combinant plusieurs matrices de même type.
- Vecteur = array 1D. Matrice = array 2D.

data.frame

- Objet 2D généré en combinant plusieurs vecteurs de n'importe quel type.
- Chaque colonne doit être du même type, mais un dataframe peut contenir des colonnes de différents types.

liste

- Une liste stocke des collections d'objets de tous types.

Vecteur

- Entité consistant en une séquence d'éléments du même type de base
- Une valeur seule est appelée une *atomic value*
- Toutes les valeurs d'un vecteur doivent avoir le **même mode** (ou classe).
 - `numeric`: seulement des nombres (e.g. `c(2, 3, 5, 3.1416)`)
 - `logical`: entrées True/False (e.g. `c(TRUE, FALSE, TRUE)`)
 - `character`: Texte, ou un mélange de texte et d'autres modes (e.g. `c("QCBS", "J'adore cet atelier")`)

Vecteur

- La création de vecteurs nécessite généralement la fonction `c`
c signifie combiner ou concaténer
- La syntaxe est:

```
vecteur <- c(valeur1, valeur2, ...)
```

Vecteur

- Vecteur numérique

```
num_vecteur <- c(1, 4, 3, 98, 32, -76, -4)
```

```
num_vecteur  
# [1] 1 4 3 98 32 -76 -4
```

- Vecteur de caractère

```
car_vecteur <- c("bleu", "rouge", "vert")
```

```
car_vecteur  
# [1] "bleu" "rouge" "vert"
```

- Vecteur logique ou booléen

```
bool_vecteur <- c(TRUE, TRUE, FALSE) # ou c(T, T, F)
```

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



Défi 7

- Créez un vecteur contenant les 5 premiers nombres impairs, à partir de 1
- Nommez-le **impair**.



Défi 7: Solution

- Créez un vecteur contenant les 5 premiers nombres impairs, à partir de 1
- Nommez-le `impair`.

```
impair <- c(1, 3, 5, 7, 9)
```

Vecteur

Astuce: Utilisez la fonction `dput` pour obtenir la structure d'un objet.

```
dput(impaire)  
# c(1, 3, 5, 7, 9)
```

Ces fonctions sont particulièrement utiles pour fournir un exemple reproductible dans le cas d'une question sur **stackoverflow** par exemple (voir application dans la partie sur les data frames) !

Vecteur

On peut utiliser des vecteurs pour faire des calculs:

```
x <- c(1:5)  
y <- 6
```

Le symbole de deux-points `:` est utilisé pour combiner toutes les valeurs entre le premier et le deuxième nombre fournis. `c(1:5)` est équivalent à `c(1, 2, 3, 4, 5)`

```
x + y  
# [1] 7 8 9 10 11  
  
x * y  
# [1] 6 12 18 24 30
```

Data frame

- Utilisé pour stocker des tables de données
- une liste de vecteurs de même longueur
- Chaque colonne = une variable
- Chaque ligne = une observation, un site, un réplicat, ...
- Différentes colonnes peuvent avoir différents modes

Data frame

Disons que vous voulez stocker ce tableau dans `R`:

site_id	pH_sol	num_sp	traitement
A1.01	5.6	17	Fertilisé
A1.02	7.3	23	Fertilisé
B1.01	4.1	15	Non fertilisé
B1.02	6.0	7	Non fertilisé

où

- "site_id" est l'identifiant du site
- "pH_sol" est la mesure du pH du sol à chaque site
- "num_sp" est le nombre d'espèces observées
- "traitement" est le traitement appliqué

Data frame

Une façon de faire est de:

- Créer des vecteurs

```
site_id <- c("A1.01", "A1.02", "B1.01", "B1.02")
pH_sol <- c(5.6, 7.3, 4.1, 6.0)
num_sp <- c(17, 23, 15, 7)
traitement <- c("Fert", "Fert", "Non_fert", "Non_fert")
```

- Puis les combiner en utilisant la fonction `data.frame`

```
mon_df <- data.frame(site_id, pH_sol, num_sp, traitement)
mon_df
#   site_id  pH_sol  num_sp  traitement
# 1 A1.01    5.6     17      Fert
# 2 A1.02    7.3     23      Fert
# 3 B1.01    4.1     15  Non_fert
# 4 B1.02    6.0      7  Non_fert
```

Nous reviendrons plus tard sur la fonction `data.frame`.

Data frame

Voici la fonction `dput` dans un exemple plus concret.

```
dput(mon_df)
```

Retourne la sortie suivante:

```
structure(list(site_id = c("A1.01", "A1.02", "B1.01", "B1.02",
), pH_sol = c(5.6, 7.3, 4.1, 6), num_sp = c(17, 23, 15, 7), traitement = c("Fert",
"Fert", "Non_fert", "Non_fert")), class = "data.frame", row.names = c(NA, -4L))
```

Data frame

Il est possible de reconstruire le data frame initial (avec certaines métadonnées associées, telles que la classe des variables par exemple) en copiant-collant la sortie précédente:

```
structure(list(site_id = c("A1.01", "A1.02", "B1.01", "B1.02"),
), pH_sol = c(5.6, 7.3, 4.1, 6), num_sp = c(17, 23, 15, 7), traitement = c("Fert",
"Fert", "Non_fert", "Non_fert")), class = "data.frame", row.names = c(NA, -4L))
#   site_id pH_sol num_sp traitement
# 1 A1.01    5.6     17      Fert
# 2 A1.02    7.3     23      Fert
# 3 B1.01    4.1     15  Non_fert
# 4 B1.02    6.0      7  Non_fert
```

Indexer un objet

- Pour regarder ou extraire une partie de nos données, on peut utiliser des **crochets** `[]`
- Nous indiquons la ou les **position(s)** des valeurs que nous voulons voir entre crochets. Il s'agit d'*indexation*.
- La façon d'indexer un objet dépend de son nombre de dimensions.
 - Pour un vecteur, il n'y a qu'une dimension: `["position"]`
 - Pour un data frame, la première position sont les "lignes" et la seconde, les "colonnes", séparé par une virgule:
`["lignes", "colonnes"]`
 - Pour indiquer la position de plusieurs éléments, nous devons placer les valeurs dans un vecteur en utilisant `c()`: `[c("ligne1":"ligne10",
c("colonne1":"colonne10"))]`.

Voyons quelques exemples!

Indexer un vecteur

On peut sélectionner une position:

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

On peut sélectionner plusieurs positions avec `c()`:

```
impair[c(2, 4)]  
# [1] 3 7
```

On peut aussi supprimer certaines valeurs à des positions particulières avec `-`:

```
impair[-c(1, 2)]  
# [1] 5 7 9
```

Indexer un objet

Si on sélectionne une position qui n'existe pas dans le vecteur, on obtient:

```
impair[c(1, 6)]  
# [1] 1 NA
```

On peut également utiliser des conditions pour sélectionner des valeurs:

```
impair[impair > 4]  
# [1] 5 7 9
```

On peut faire la même chose avec un vecteur de texte

```
car_vecteur  
# [1] "bleu" "rouge" "vert"  
car_vecteur[car_vecteur == "bleu"]  
# [1] "bleu"
```

Les déclarations logiques telles que `>` seront décrites en détail plus loin



Défi 8

À l'aide du vecteur `num_vecteur`

- Extraire la 4ème valeur
- Extraire les 1ère et 3ème valeurs
- Extraire toutes les valeurs sauf la 2ème et la 4ème



Défi 8: Solution

- Extraire la 4ème valeur

```
num_vecteur[4]  
# [1] 98
```

- Extraire les 1ère et 3ème valeurs

```
num_vecteur[c(1, 3)]  
# [1] 1 3
```

- Extraire toutes les valeurs sauf la 2ème et la 4ème

```
num_vecteur[c(-2, -4)]  
# [1] 1 3 32 -76 -4
```



Défi 9

Explorez la différence entre ces 2 lignes de code:

```
car_vecteur == "bleu"
```

```
car_vecteur[car_vecteur == "bleu"]
```



Défi 9: Solution

```
car_vecteur == "bleu"  
# [1] TRUE FALSE FALSE
```

Dans cette ligne de code, vous **testez une déclaration logique**. Pour chaque entrée dans le `car_vecteur`, R vérifie si l'entrée est égale à `bleu` ou non.

```
car_vecteur[car_vecteur == "bleu"]  
# [1] "bleu"
```

Dans cette ligne de code, vous demandez à R d'extraire toutes les valeurs du vecteur `car_vecteur` qui sont exactement égales à `bleu`.

Indexer un data frame

Pour indexer un **data frame**, vous devez spécifier deux dimensions: numéro de ligne et de colonne, en utilisant la syntaxe suivante:

```
nom_du_data_frame[ligne, colonne]
```

Indexer un data frame: exemples

`mon_df[1,]` → Extrait la 1ère ligne

Notez qu'un index vide sélectionne **toutes les valeurs**.

`mon_df[, 3]` → Extrait la 3ème colonne

`mon_df[2, 4]` → Extrait la 2ème ligne de la 4ème colonne

Indexer un data.frame: exemples

`mon_df[c(2, 4),]` → Extrait les 2ème et 4ème lignes

Les exemples donnés jusqu'ici sont également valables pour indexer les matrices, ce n'est pas le cas pour les exemples suivants.

`mon_df$site_id` → Extrait la variable `site_id` du data frame

`mon_df$site_id[2]` → Extrait la 2ème valeur de la variable `site_id` du data frame

`mon_df[c("site_id", "pH_sol")]` → Extrait les variables `site_id` et `pH_sol` du data frame.

Note sur les déclarations logiques

R permet de tester les déclarations logiques, i.e. tester si une déclaration est vraie ou fausse.

Vous devez utiliser des opérateurs logiques pour cela.

Opérateur	Description	Exemple
<code><</code> et <code>></code>	inférieur ou supérieur à	<code>impair > 3</code>
<code><=</code> et <code>>=</code>	inférieur/supérieur ou égale à	<code>impair >= 3</code>
<code>==</code>	égale à	<code>impair == 3</code>
<code>!=</code>	non égale à	<code>impair != 3</code>
<code>x y</code>	x OU y	<code>impair[impair >= 5 impair < 3]</code>
<code>x & y</code>	x ET y	<code>impair[impair >=3 & impair < 7]</code>



Défi 10

1. Extrayez la colonne `num_sp` de `mon_df` et multipliez ses valeurs par les premières quatre valeurs de `num_vecteur`.
2. Ensuite, écrivez une déclaration logique qui vérifie si les valeurs obtenues sont supérieures à 25.



Défi 10: Solution

- Extrayez la colonne `num_sp` de `mon_df` et multipliez ses valeurs par les premières quatre valeurs de `num_vecteur`.

```
mon_df$num_sp * num_vecteur[c(1:4)]  
# [1] 17 92 45 686  
# ou encore  
mon_df[, 3] * num_vecteur[c(1:4)]  
# [1] 17 92 45 686
```

- Ensuite, écrivez une déclaration logique qui vérifie si les valeurs obtenues sont supérieures à 25.

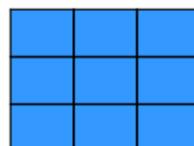
```
(mon_df$num_sp * num_vecteur[c(1:4)]) > 25  
# [1] FALSE TRUE TRUE TRUE
```

Matrice, Array et Liste

- Dans cet atelier, nous nous sommes principalement attardés aux **vecteurs** et aux **data frames**.
- Dans le futur, il est fort possible que vous ayez à travailler avec des **matrices**, des **listes** ou des **arrays**.
- La façon d'accéder à l'information diffère selon la dimension et le type de l'objet.



Vecteur

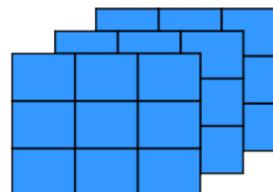


Matrice



Data frame

*Les colonnes peuvent être de différentes classes



Array

Liste {
Vecteur
Matrice
Data frame
Array}

3. Manipuler les objets dans R

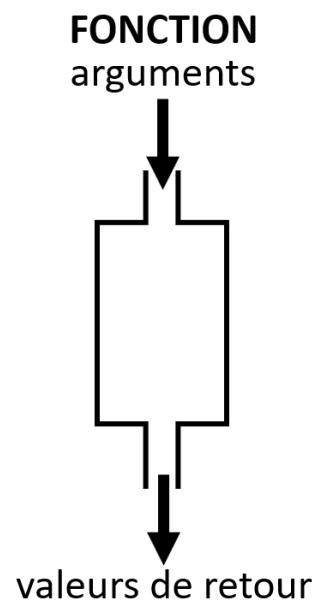
Fonction

Fonction

Une fonction est un outil pour simplifier votre vie.

Elle vous permet d'exécuter rapidement des opérations sur des objets sans avoir à écrire toutes les étapes mathématiques.

Une fonction nécessite des valeurs d'entrée appelées **arguments** (ou paramètres). Elle effectue ensuite des opérations cachées en utilisant ces arguments et donne une **valeur de retour**.



Fonction

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

```
nom_fonction(argument1, argument2)
```

Argument

Les arguments sont des **valeurs** et des **instructions** nécessaires à la fonction pour être exécutée.

Les objets stockant ces valeurs et ces instructions peuvent être utilisés dans des fonctions. Par exemple:

```
a <- 3  
b <- 5  
sum(a, b)  
# [1] 8
```



Défi 11

- Créez un vecteur `a` qui contient tous les nombres de 1 à 5
- Créez un objet `b` qui a une valeur de 2
- Additionnez `a` et `b` en utilisant l'opérateur de base `+` et enregistrez le résultat dans un objet appelé `resultat_add`
- Additionnez `a` et `b` ensemble en utilisant la fonction `sum` et enregistrez le résultat dans un objet appelé `resultat_sum`
- Comparez `resultat_add` et `resultat_sum`. Sont-ils différents?
- Additionnez 5 à `resultat_sum` en utilisant la fonction `sum`



Défi 11: Solution

```
a <- c(1:5)
b <- 2

resultat_add <- a + b

resultat_sum <- sum(a, b)

resultat_add
# [1] 3 4 5 6 7

resultat_sum
# [1] 17

sum(resultat_sum, 5)
# [1] 22
```

L'opération `+` sur le vecteur `a` ajoute 2 à chaque élément. Le résultat est un vecteur.

La fonction `sum` concatène toutes les valeurs fournies puis les somme. C'est la même chose que de faire $1+2+3+4+5+2$.

Argument

- Les arguments ont chacun un **nom** pouvant être spécifié lors d'un appel de fonction.
- Si le nom n'est pas spécifié, l'ordre des arguments est important.
- Si le nom est spécifié, l'ordre n'a pas d'importance.



Défi 12

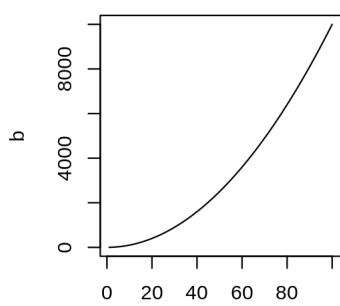
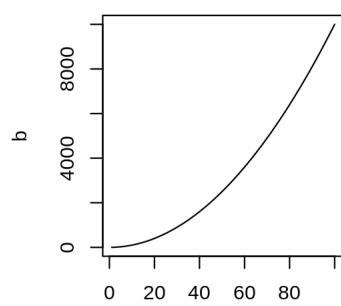
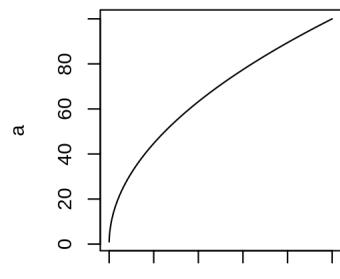
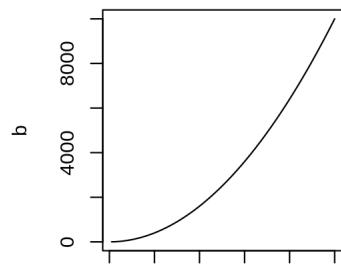
`plot` est une fonction qui dessine un graphique de `y` en fonction de `x`. Il nécessite deux noms d'argument `x` et `y`. Quelles sont les différences entre les lignes suivantes?

```
a <- 1:100  
b <- a^2  
plot(a, b, type = "l")  
plot(b, a, type = "l")  
plot(x = a, y = b, type = "l")  
plot(y = b, x = a, type = "l")
```

L'argument `type` de la fonction `plot` permet de choisir le type de graphique. Essayez la même fonction sans cet argument.

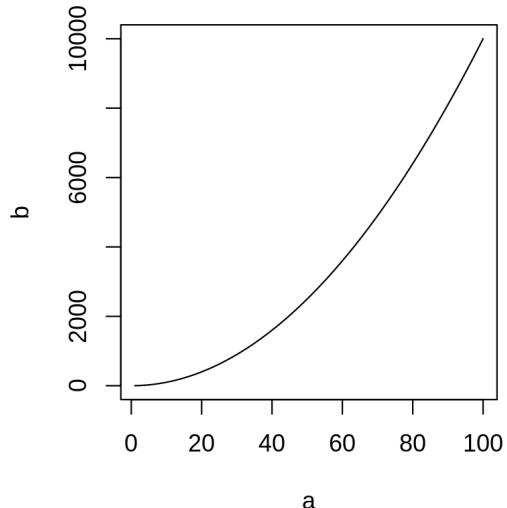
Défi 12: Solution

`plot` est une fonction qui dessine un graphique de y en fonction de x . Il nécessite deux noms d'argument `x` et `y`. Quelles sont les différences entre les lignes suivantes?

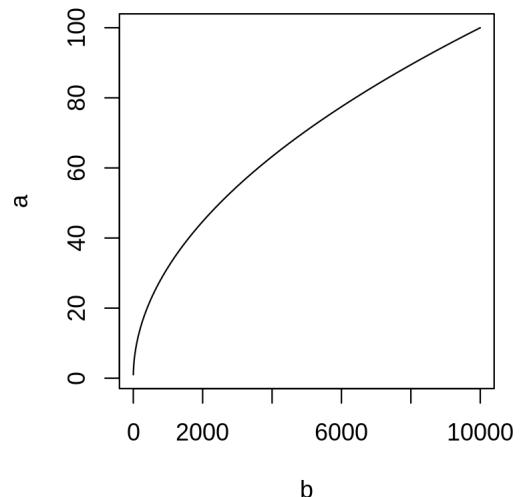


Défi 12: Solution

```
plot(a, b, type = "l")
```



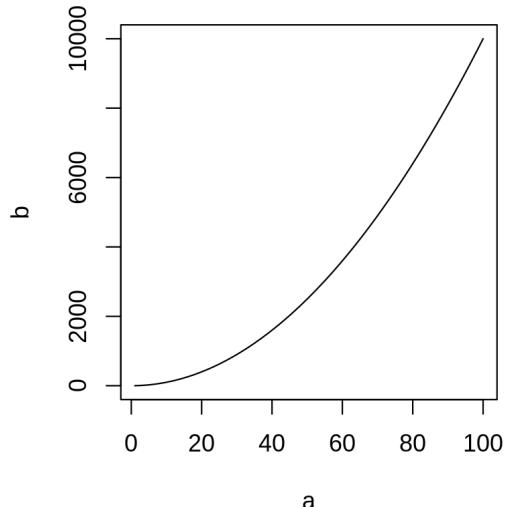
```
plot(b, a, type = "l")
```



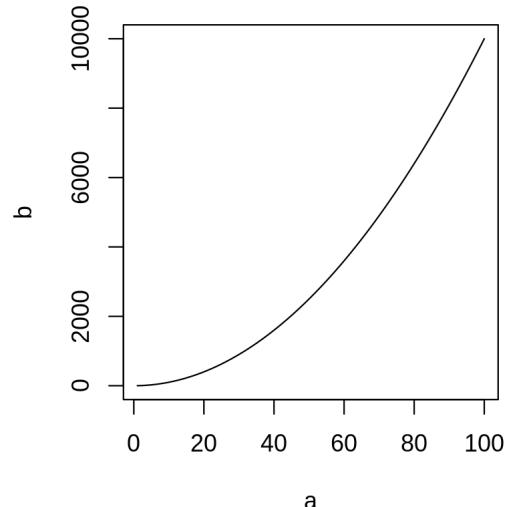
La forme du graphe change, car nous n'avons pas spécifié le nom des arguments, alors l'ordre devient important.

Défi 12: Solution

```
plot(x = a, y = b, type = "l")
```



```
plot(y = b, x = a, type = "l")
```



Idem que `plot(a, b)`. Le nom des arguments est spécifié, l'ordre n'est pas important.

4. Installer et utiliser les packages R

Package

Package

Les *packages* sont un **regroupement de fonctions** et/ou **de données** partageant un thème **similaire**, par ex. statistiques, analyse spatiale, graphique, etc.

Tout le monde peut développer des *packages* et les mettre à la disposition des autres.

Beaucoup de packages disponibles via le *Comprehensive R Archive Network* ou **CRAN** et maintenant encore plus via GitHub [GitHub](#).

Devinez combien de *packages* sont disponibles aujourd'hui?

Combien de packages

R Package Documentation

A comprehensive index of R packages and documentation from CRAN, Bioconductor, GitHub and R-Forge.

Search for anything R related

Find an R package by name, find package documentation, find R documentation, find R functions, search R source code...

Search

16732

CRAN PACKAGES

1903

BIOCONDUCTOR PACKAGES

2130

R-FORGE PACKAGES

56786

GITHUB PACKAGES

Find an R package

Browse R language docs

Run R code online

Over 9,000 packages are preinstalled!

Create an R Notebook

rdrr.io consulté le 20 mars 2020

Package

Pour installer des *packages* sur votre ordinateur, utilisez la fonction `install.packages`.

```
install.packages("nom du package")
```

L'installation d'un *package* ne suffit pas pour l'utiliser. Vous devez aussi le charger dans votre espace de travail à l'aide de la fonction `library` pour pouvoir l'utiliser.

```
library(nom du package)
```

Package: exemple

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

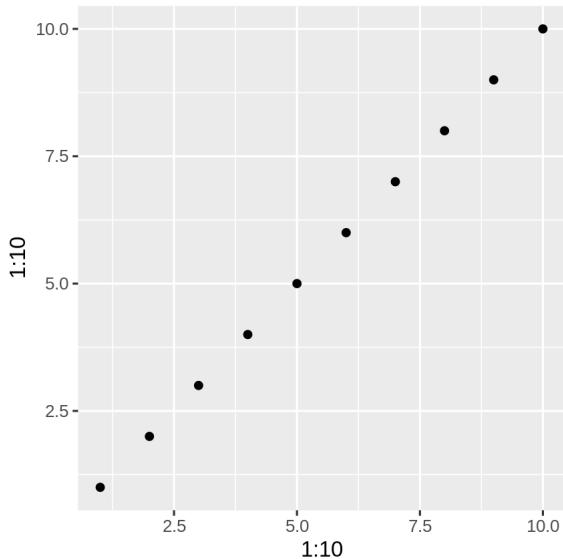
```
Installing package into '/home/lab0/R/x86_64-redhat-linux-gnu-library/3.3'  
(as 'lib' is unspecified)
```

```
qplot(1:10, 1:10)
```

```
## Error: could not find function "qplot"
```

Package: exemple

```
library(ggplot2)  
qplot(1:10, 1:10)
```



Pour plus de détails sur le package `ggplot2`, voir l'atelier #3 - **Introduction à ggplot2 !!!**

4. Installer et utiliser les packages R

Obtenir de l'aide

Recherche de fonctions

WOW! R est tellement génial! Tellement de fonctions incroyablement utiles!

Mais... comment puis-je les trouver?

Pour trouver une fonction qui fait une action spécifique dans vos *packages* installés, vous pouvez utiliser ?? suivi d'un terme de recherche.

Disons que nous voulons créer une **séquence** de nombres impairs compris entre 0 et 10, comme nous l'avons fait précédemment. Nous pouvons rechercher dans nos *packages* toutes les fonctions qui contiennent le mot **séquence** dans leur description:

```
??sequence
```

Notez que toutes les recherches doivent se faire en anglais

Résultats de la recherche

Vignettes:

[colorspace::hcl-colors](#) HCL-Based Color Palettes in R

Code demonstrations:

[foreach::sincSEQ](#) computation of the sinc function

Help pages:

adegraphics::sortparamADEg	Sort a sequence of graphical parameters
ape::AAbin	Amino Acid Sequences
ape::DNAbin	Manipulate DNA Sequences in Bit-Level Format
ape::alview	Print DNA or AA Sequence Alignement
ape::tree.build	Internal Ape Functions
ape::as.alignment	Conversion Among DNA Sequence Internal Formats
ape::base.freq	Base frequencies from DNA Sequences
ape::clustal	Multiple Sequence Alignment with External Applications
ape::del.gaps	Delete Alignment Gaps in DNA Sequences
●	
●	
●	
xts::timeBasedSeq	Create a Sequence or Range of Times
base::seq.Date	Generate Regular Sequences of Dates
base::seq.POSIXt	Generate Regular Sequences of Times
base::seq	Sequence Generation
base::sequence	Create A Vector of Sequences
Matrix::abIndex-class	Class "abIndex" of Abstract Index Vectors
Matrix::abIndex	Sequence Generation of "abIndex", Abstract Index Vectors
Matrix::solve	Methods in Package Matrix for Function 'solve()'
Matrix::sparseQR-class	Sparse QR decomposition of a sparse matrix

Résultats de la recherche

Vignettes:

Nom de la fonction

[colorspace::hcl-colors](#)

Description

HCL-Based Color Palettes in R

Code demonstrations:

[foreach::sincSEQ](#)

computation of the sinc function

Help pages:

[adegraphics::sortparamADEg](#)

Sort a sequence of graphical parameters

[ape::AAbin](#)

Amino Acid Sequences

[ape::DNAbin](#)

Manipulate DNA Sequences in Bit-Level Format

[ape::alview](#)

Print DNA or AA Sequence Alignement

[ape::tree.build](#)

Internal Ape Functions

[ape::as.alignment](#)

Conversion Among DNA Sequence Internal Formats

[ape::base.freq](#)

Base frequencies from DNA Sequences

[ape::clustal](#)

Multiple Sequence Alignment with External Applications

[ape::del.gaps](#)

Delete Alignment Gaps in DNA Sequences

-
-
-

nom_package::fonction

[xts::timeBasedSeq](#)

Create a Sequence or Range of Times

[base::seq.Date](#)

Generate Regular Sequences of Dates

[base::seq.POSIXt](#)

Generate Regular Sequences of Times

[base::seq](#)

Sequence Generation

[base::sequence](#)

Create A Vector of Sequences

[Matrix::abIndex-class](#)

Class "abIndex" of Abstract Index Vectors

[Matrix::abIndex](#)

Sequence Generation of "abIndex", Abstract Index Vectors

[Matrix::solve](#)

Methods in Package Matrix for Function 'solve()'

[Matrix::sparseQR-class](#)

Sparse QR decomposition of a sparse matrix

Résultats de la recherche

Vignettes:

Nom de la fonction

[colorspace::hcl-colors](#)

Description

HCL-Based Color Palettes in R

Code demonstrations:

[foreach::sincSEQ](#)

computation of the sinc function

Help pages:

[adegraphics::sortparamADEg](#)

Sort a sequence of graphical parameters

[ape::AAbin](#)

Amino Acid Sequences

[ape::DNAbin](#)

Manipulate DNA Sequences in Bit-Level Format

[ape::alview](#)

Print DNA or AA Sequence Alignement

[ape::tree.build](#)

Internal Ape Functions

[ape::as.alignment](#)

Conversion Among DNA Sequence Internal Formats

[ape::base.freq](#)

Base frequencies from DNA Sequences

[ape::clustal](#)

Multiple Sequence Alignment with External Applications

[ape::del.gaps](#)

Delete Alignment Gaps in DNA Sequences



nom_package::fonction

[xts::timeBasedSeq](#)

Create a Sequence or Range of Times

[base::seq.Date](#)

Generate Regular Sequences of Dates

[base::seq.POSIXt](#)

Generate Regular Sequences of Times

[base::seq](#)

Sequence Generation

[base::sequence](#)

Create A Vector of Sequences

[Matrix::abIndex-class](#)

Class "abIndex" of Abstract Index Vectors

[Matrix::abIndex](#)

Sequence Generation of "abIndex", Abstract Index Vectors

[Matrix::solve](#)

Methods in Package Matrix for Function 'solve()'

[Matrix::sparseQR-class](#)

Sparse QR decomposition of a sparse matrix

Aide pour les fonctions

OK! Alors utilisons la fonction `seq` !

Mais attendez ... comment ça marche? De quels arguments la fonction a-t-elle besoin?

Pour trouver des informations sur une fonction en particulier, utilisez `?`

`?seq`

Page d'aide

seq {base}

R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

... arguments passed to or from methods.

from, to the starting and (maximal) end values of the sequence. Of length 1 unless just `from` is supplied as an unnamed argument.

by number: increment of the sequence.

length.out desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.

along.with take the length from the length of this argument.

Details

Numerical inputs should all be [finite](#) (that is, not infinite, [NaN](#) or NA).

The interpretation of the unnamed arguments of `seq` and `seq.int` is *not* standard, and it is recommended always to name the arguments when programming.

Description

- `nom_fonction {nom_packa}`
- `Description`: une courte description de ce que peut faire la fonction.

`seq {base}`

R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

- Comment appeler la fonction
- Si `name = value` est présent, une valeur par défaut est fournie si l'argument est manquant. L'argument devient facultatif.
- Autres fonctions connexes décrites dans la page d'aide

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

- Description de tous les arguments et de leur utilisation

Arguments

`...` arguments passed to or from methods.

`from, to` the starting and (maximal) end values of the sequence. Of length 1 unless just `from` is supplied as an unnamed argument.

`by` number: increment of the sequence.

`length.out` desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.

`along.with` take the length from the length of this argument.

Détails

- Une description détaillée du fonctionnement des fonctions et de leurs caractéristiques

Valeur de retour, Voir aussi, Exemples

- Une description de la valeur de retour

Value

`seq.int` and the default method of `seq` for numeric arguments return a vector of type "integer" or "double": programmers should not rely on which.

`seq_along` and `seq_len` return an integer vector, unless it is a *long vector* when it will be double.

- Autres fonctions connexes pouvant être utiles

See Also

The methods [seq.Date](#) and [seq.POSIXt](#).

[:](#), [rep](#), [sequence](#), [row](#), [col](#).

- Exemples reproductibles

Examples

```
seq(0, 1, length.out = 11)
seq(stats:::rnorm(20)) # effectively 'along'
seq(1, 9, by = 2)      # matches 'end'
seq(1, 9, by = pi)     # stays below 'end'
seq(1, 6, by = 3)
seq(1.575, 5.125, by = 0.05)
seq(17) # same as 1:17, or even better seq_len(17)
```

Défi 13



1. Créez une séquence de nombres pairs de 0 à 10 en utilisant la fonction `seq`.
2. Créez un vecteur non ordonné de vos nombres favoris, puis triez votre vecteur dans l'ordre inverse.



Défi 13: Solutions

1. Créez une séquence de nombres pairs de 0 à 10 en utilisant la fonction `seq`.

```
seq(from = 0, to = 10, by = 2)  
# [1] 0 2 4 6 8 10
```

```
seq(0, 10, 2)  
# [1] 0 2 4 6 8 10
```

1. Créez un vecteur non ordonné de vos nombres favoris, puis triez votre vecteur dans l'ordre inverse.

```
numbers <- c(2, 4, 22, 6, 26)  
sort(numbers, decreasing = T)  
# [1] 26 22 6 4 2
```

Autres façons d'obtenir de l'aide

Généralement, votre meilleure source d'information sera votre moteur de recherche préféré!

Voici quelques conseils pour l'utiliser efficacement:

- Faites vos recherches en anglais
- Utilisez le mot-clé **R** au début de votre recherche
- Définissez précisément ce que vous recherchez
- Apprenez à lire les forums de discussion, par exemple [StackOverflow](#). Il est fort probable que d'autres personnes aient déjà eu le même problème que vous et aient posé la question à ce sujet.
- N'hésitez pas à chercher à nouveau en utilisant différents mots-clés!

Défi 14



Recherchez les fonctions appropriées pour effectuer les opérations suivantes:

- Racine carrée
- Moyenne de plusieurs nombres
- Combiner deux data frames par colonnes
- Lister des objets disponibles dans votre espace de travail

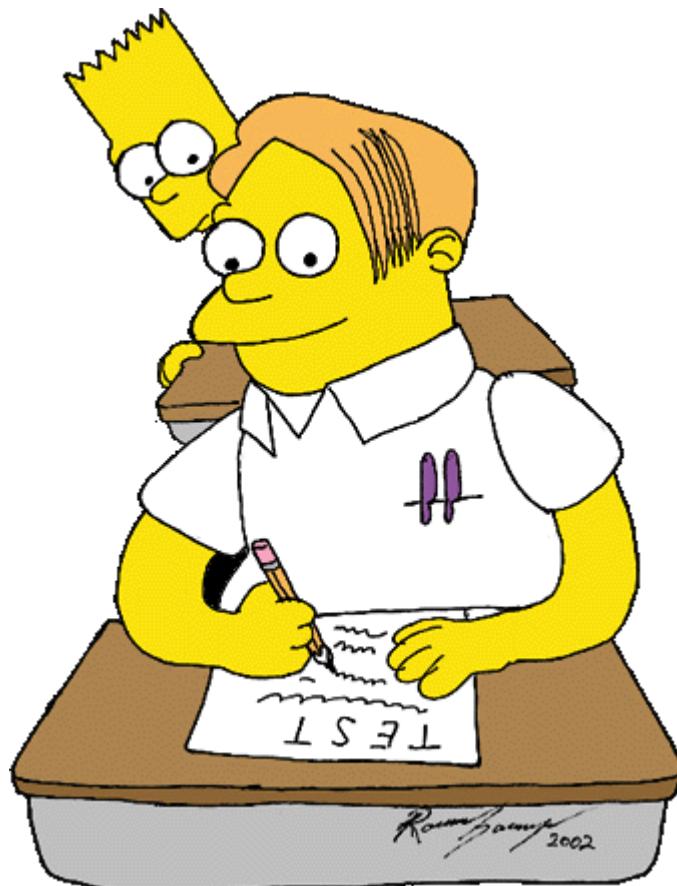


Défi 14: Solutions

- Racine carrée
 - `sqrt`
- Moyenne de plusieurs nombres
 - `mean`
- Combiner deux data frames par colonnes
 - `cbind`
- Lister des objets disponibles dans votre espace de travail
 - `ls`

Ressources additionnelles

Cheat 4ever

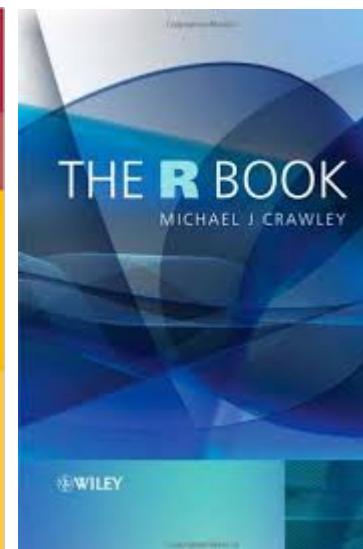
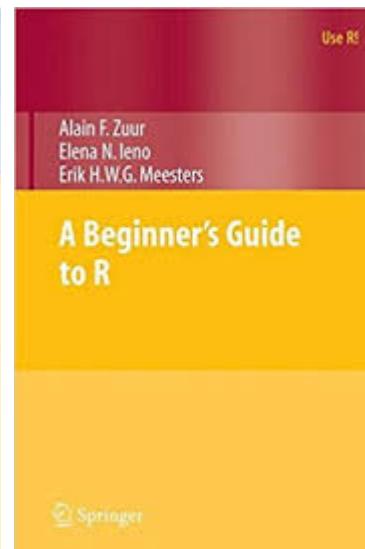
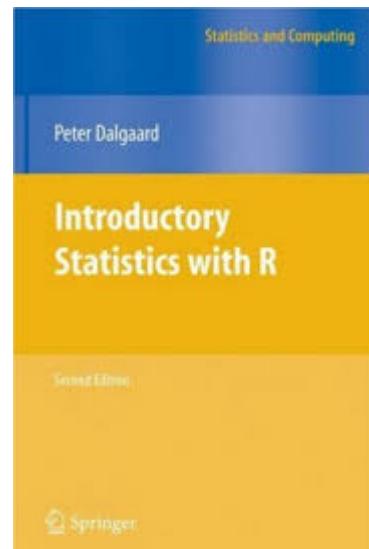
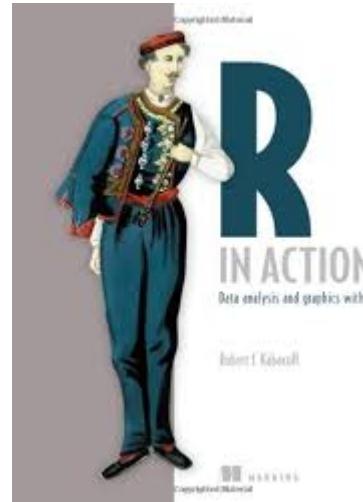
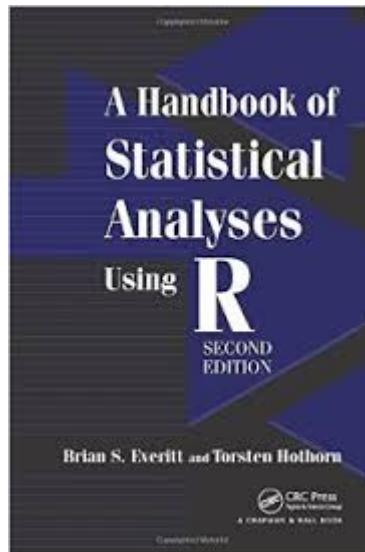
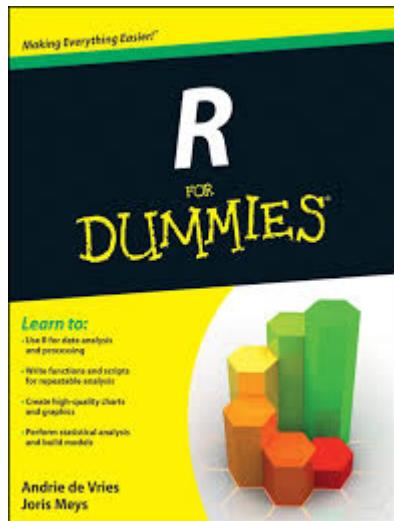


De nombreuses **cheat sheets** sont disponibles en ligne.

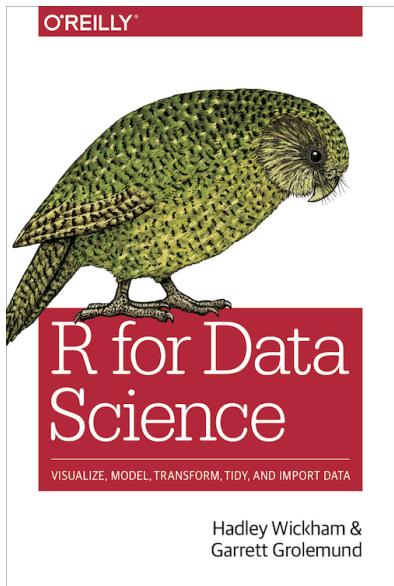
Ouvrez-les directement depuis **Rstudio**: Help → Cheatsheets

Cheat 4ever

Quelques livres utiles



Quelques sites internet utiles



Cookbook for R

An Introduction to R

R Reference card 2.0

Merci d'avoir participé!

