

# Atelier 1: RStudio

Pascal Brissette (U. McGill)

1/1/23



Figure 1: Cr dit photographique: Matthieu Comoy, Unsplash

**Cet atelier propose une initiation   l'environnement RStudio et   la structure de donn es de base de R, le vecteur. Il est destin  aux  tudiant e s et chercheur euse s en sciences humaines sans formation particuli re en programmation.**


## RStudio: environnement de travail et d'expérimentation

Votre lieu de travail s'appelle RStudio ou [Posit](#). Il s'agit d'un véritable terrain de jeu et d'expérimentation. Pour pouvoir l'utiliser, vous devrez au préalable installer la dernière version du langage [R](#) et, bien sûr, de [RStudio](#). Comme les raccourcis peuvent varier selon le système d'exploitation de l'ordinateur, nous allons, par souci de simplicité, utiliser la version "Cloud" de RStudio, [Posit-Cloud](#). Les ateliers pourront être importés directement de GitHub à travers Posit-Cloud.

### Qu'est-ce qu'un projet R?

Si vous créez un script où, par exemple, vous demandez à R de lire un jeu de données contenu dans un répertoire de votre ordinateur, le chemin utilisé par R pour lire ces données est fixé à partir de la racine de votre ordinateur. Si vous transmettez votre script et vos données à un collègue pour qu'il reproduise les résultats et vous donne son avis, ce dernier devra modifier le chemin d'accès aux données parce que celles-ci se trouvent désormais sur son ordinateur. Cela est peut rendre laborieux la reproduction des flux de travail.

La création préalable d'un "projet R" permet notamment d'éviter ces changements de chemins et rend possible l'exécution des scripts où quel que soit l'environnement où ils sont placés. Un "projet R" se présente comme un conteneur où vous ferez vos expérimentations et déposerez vos scripts et données. Lorsque viendra le temps de partager votre travail, c'est le conteneur au complet que vous déplacerez et son ouverture dans un autre environnement se fera sans qu'il soit nécessaire de modifier le chemin ni d'installer les extensions qui ont été utilisées lors de la création du projet.

Pour créer un "projet R", vous pouvez cliquer sur l'icône  située dans la barre d'outils. Vous pouvez également utiliser le menu principal File/New Project... Vous trouverez plus de détails sur l'utilisation des projets R et de bons conseils sur la structuration des dossiers à l'intérieur d'un "projet R" dans [cet article](#) de Martin Chan.

### Exécuter ou ne pas exécuter les commandes...

Dans un script R, les caractères qui suivent un croisillon # ne sont pas exécutés. Vous pouvez donc vous servir de ce symbole pour insérer des titres ou des commentaires dans votre script.

Il est toujours recommandé de commenter votre code, d'indiquer ce que les lignes de commandes sont censées produire, ce que vous souhaitez faire. Ces commentaires vous seront utiles lorsque, quelques jours, semaines ou mois plus tard, vous reprendrez le travail ou souhaitez vous en inspirer pour créer une autre tâche. En cas de pépin ou de blocage ("troubleshooting"), ces commentaires indiqueront aux personnes qui vous aident quelle logique vous avez suivie.

```
# La présente ligne ne sera pas exécutée, car elle est précédée du croisillon.  
  
list.files("donnees") #La commande précédente sera exécutée, mais non ce commentaire.
```

```
[1] "actes_criminels.csv"           "fichiers_xls"  
[3] "filmographies_iconv.csv"       "filmographies_realisatrices_qc.tsv"  
[5] "maria_chapdelaine.txt"         "maria_freq_brute_pretendants.csv"  
[7] "maria_freq_pretendants_chapitre.csv" "ouvrages.csv"  
[9] "ouvrages.RDS"                 "publicationsqc.csv"  
[11] "rc_longes_metrages.csv"        "scouine.epub"
```

```
# Que semble produire la fonction que vous venez d'exécuter? Dans quel contexte pourriez-vous l'utiliser?
```

Dans un script R de base, le résultat de la commande n'apparaît que dans la fenêtre “Console”. Dans un document Quarto comme celui-ci, le résultat apparaît également sous les lignes de code (“code chunk”).

Pour exécuter une commande, vous pouvez, dans un document Quarto, cliquer sur la flèche à la droite de la fenêtre de code ou encore mettre votre curseur au début de la ligne de code et utiliser le raccourci **Command+Enter**.

```
2 * 2 # Exécuter cette ligne de code avec le raccourci.
```

```
[1] 4
```

## Obtenir de l'aide

À tout moment, sans même sortir de l'environnement de travail, on peut obtenir de l'aide sur une fonction en particulier ou sur une extension (“package”).

## Documentation sur les extensions

Pour illustrer le travail avec les extensions, nous utiliserons **proustr**, créé par Colin Fay en 2017. Cette extension contient les sept romans composant *La Recherche du temps perdu* de Marcel Proust, ainsi que la liste de tous les personnages de la *Recherche* et des fonctions permettant par exemple de choisir des citations au hasard.

On peut obtenir de l'aide sur une extension une fois qu'elle a été installée (si elle ne l'est pas déjà) et activée (ce sont des opérations distinctes). Pour installer une extension, on utilise la fonction `install.packages("...")`, et pour activer la même extension, on utilise

la fonction `library()`. Quant à la documentation de l’extension, elle sera appelée par la fonction `help(package="...")`. Elle apparaîtra dans la fenêtre inférieure droite de RStudio, sous l’onglet “Help”.

**Note:** pour éviter les messages d’erreur, n’oubliez pas de mettre le nom de l’extension entre guillemets lorsque vous l’installez. Une fois installée, cette extension devient un objet et peut donc être appelée par son nom d’objet, sans guillemets.

**Rappel:** vous trouverez toutes les extensions approuvées par l’équipe de RStudio sur [CRAN](https://cran.r-project.org/) (“Comprehensive R Archive Network”). Il y en a plus de 18 000 en ce mois d’août 2022.

```
# install.packages("proustr") # Installation d'une extension

library(proustr) # Activation d'une extension

help(package="proustr")

??proustr # Ce raccourci produit un résultat similaire.
```

Disons-le tout de suite: la documentation a un aspect assez austère. Lorsque vous maîtriserez bien vos outils, vous apprécierez ces présentations directes, sans flafra. Pour un premier contact avec une extension, il vaut mieux consulter les vignettes ou les aide-mémoire (“cheat sheet”). Les liens vers les vignettes se trouvent en tête de la documentation. Quant aux aide-mémoire, votre moteur de recherche préféré vous y conduira aisément (exemple de recherche: “proustr cheat sheet”).

Pour obtenir de l’aide sur une fonction en particulier, on écrit son nom précédé d’un point d’interrogation.

```
?proust_random()

proust_random(count = 2) # Essayez cette fonction. Que renvoie-t-elle selon vous?
```

```
[1] "Et vous me remercieriez, car, si je suis expert en fait de mariages, je ne le suis pas m
```

## Créer ses premiers objets

Dans R, tout est un objet dont on peut voir et explorer le contenu, qu’on peut manipuler et modifier. Les fonctions que vous créerez, les calculs que vous ferez, les graphiques que vous produirez, tout pourra être encapsulé dans des “conteneurs” et devenir un objet nommé. À tout moment, vous pouvez obtenir la liste de vos objets avec la fonction `ls()`. Dans

RStudio, ces derniers sont rassemblés dans la fenêtre supérieure droite, sous l'onglet (par défaut) "Environnement".

Pour créer un objet, on utilise l'opérateur `<-`. Cet opérateur prend la forme d'une flèche; il associe à une variable le résultat de la ligne de commande qui se trouve à sa droite.

```
# La fonction proust_random() renvoie une citation aléatoire
# tirée de La Recherche du temps perdu de Proust.

citation_proust_1 <- proust_random(2) # Création d'un premier objet
citation_proust_2 <- proust_random(5) # Création d'un deuxième objet

ls()
```

```
[1] "citation_proust_1" "citation_proust_2"
```

```
# Par convention, on utilise l'opérateur <- pour les assignations et le symbole `=` pour d
```

Pour composer le nom d'un objet et pouvoir l'inscrire dans l'environnement de travail, on peut utiliser un mélange de lettres et de chiffres. Il faut cependant éviter d'utiliser:

- un chiffre en début de ligne;
- des lettres accentuées et des espaces;
- des noms de fonctions existantes (ex.: `length`, `dim`, `apply`), soit des mots réservés.

Enfin, vous utiliserez idéalement des noms qui traduisent au mieux le contenu de l'objet.

## Les principaux types d'objets

Les deux objets que nous avons créés dans la section précédente sont des vecteurs de longueur 1. Il ne faut pas confondre la longueur de la chaîne de caractères, c'est-à-dire le nombre de caractères que contient une chaîne, et la longueur d'un objet, soit le nombre d'éléments dont il est constitué. Le nombre de caractères d'une chaîne peut être obtenue avec la fonction `nchar()`, tandis que la longueur de l'objet est obtenue avec `length()`.

```
nchar(citation_proust_1)
```

```
[1] 389
```

```
length(citation_proust_1)
```

```
[1] 1
```

Le vecteur est l'unité de base de R, la structure de données élémentaire. Toutes les autres sont construites à partir de vecteurs. Il est donc essentiel de savoir les explorer et les manipuler.

Le vecteur peut être comparé à un dossier (une “chemise”) qui ne pourrait contenir qu'un seul type de document (des photos OU des documents écrits OU des feuilles de calcul, etc.). Il s'agit d'une structure de données homogène ou, pour utiliser le vocabulaire courant, « atomique ». Si vous essayez de concaténer des données de divers types, c'est-à-dire d'en faire un vecteur, R forcera leur homogénéisation. Les deux objets que nous avons créés avec la fonction `proust_random()` sont de type `character`. Cependant,

```
mode(citation_proust_1)
```

```
[1] "character"
```

On peut assembler des vecteurs de même type avec la fonction `c()` (“c” pour “concatenate”) et les assigner à un nouvel objet:

```
citation_proust1_2 <- c(citation_proust_1, citation_proust_2)
```

```
print(citation_proust1_2) # Observez le résultat
```

```
[1] "Car une œuvre, même de confession directe, est pour le moins intercalée entre plusieurs  
[2] "Les républicains les plus sages pensaient qu'il était fou de faire la séparation de l'É
```

```
length(citation_proust1_2)
```

```
[1] 2
```

La fonction `length()` indique que le nouvel objet comprend deux éléments. Ceux-ci peuvent être isolés ou extraits à l'aide des crochets `[]`, placé immédiatement après le nom de l'objet. On indiquera dans le crochet l'indice de l'élément qu'on souhaite observer ou manipuler.

```
citation_proust1_2[2] # Extraction du deuxième élément du vecteur
```

```
[1] "Les républicains les plus sages pensaient qu'il était fou de faire la séparation de l'É"
```

On peut ajouter au vecteur autant d'éléments que l'on souhaite en utilisant la fonction `append()` ou `c()`.

```
length(citation_proust1_2) # Vérification du nombre d'éléments du vecteur
```

```
[1] 2
```

```
citation3 <- proust_random(1) # Nouvelle citation
citation4 <- proust_random(1) # Nouvelle citation

citation_proust1_2 <- c(citation_proust1_2, citation3) # Combinaison 1

length(citation_proust1_2)
```

```
[1] 3
```

```
citation_proust1_2 <- append(citation_proust1_2, citation4)

length(citation_proust1_2)
```

```
[1] 4
```

On peut attribuer des noms aux éléments d'un vecteur. Cela permettra ultérieurement d'extraire l'élément à partir de ce nom au lieu d'avoir à utiliser son indice numérique (position dans le vecteur). On dira qu'un vecteur possède un attribut `names`, attribut qu'on peut définir avec la fonction `names()`.

```
names(citation_proust1_2) <- c("Citation1", "Citation2", "Citation3", "Citation4") # Attribuer des noms

str(citation_proust1_2) # La fonction str() indique que les éléments du vecteur
```

```
Named chr [1:4] "Car une œuvre, même de confession directe, est pour le moins intercalée en
- attr(*, "names")= chr [1:4] "Citation1" "Citation2" "Citation3" "Citation4"
```



```
# sont pourvus d'un attribut "names"

citation_proutst1_2["Citation1"] # Extraction de l'élément grâce au nom
```

"Car une œuvre, même de confession directe, est pour le moins intercalée entre plusieurs épi.

Si on souhaite éliminer un élément du vecteur, on peut l'isoler avec son indice ou son nom, et lui assigner la valeur NULL. Si on veut plutôt en modifier le contenu, on l'isole avec son indice ou son nom, puis on lui assigne un nouveau contenu (l'écrasement du contenu antérieur est automatique).

```
citation_proutst1_2[1] <- proust_random(1)

citation_proutst1_2["Citation1"] <- "test"

citation_proutst1_2 <- citation_proutst1_2[c(2,3)]

citation_proutst1_2[-c(2)]
```

"Les républicains les plus sages pensaient qu'il était fou de faire la séparation de l'Église

R contient, dès son chargement, une multitude de vecteurs prêts à être utilisés. L'objet `letters`, par exemple, est un vecteur contenant 26 éléments (je vous laisse deviner lesquels).

```
mes_lettres <- letters
length(mes_lettres)
```

```
[1] 26
```

```
identical(mes_lettres, letters) # la fonction `identical()` vérifie l'identité entre les o
```

```
[1] TRUE
```

Chaque lettre du vecteur `letters` est isolée des autres, comme le montrent les guillemets encadrant chaque élément. On pourrait — et c'est une chose que vous ferez sur une base régulière — assembler ces éléments en une seule chaîne de caractères. Cela se fait avec la



fonction `paste()`. Cette fonction prend comme arguments une série de vecteurs à assembler (...), un séparateur entre les vecteurs à assembler `sep =`, ainsi qu'une option d'amalgame des éléments de ces vecteurs `collapse =`. Essayez cette fonction avec le vecteur `letters` ou avec `citation_proust1_2`.

```
?paste
paste(letters,letters, sep = "---")
```

```
[1] "a---a" "b---b" "c---c" "d---d" "e---e" "f---f" "g---g" "h---h" "i---i"
[10] "j---j" "k---k" "l---l" "m---m" "n---n" "o---o" "p---p" "q---q" "r---r"
[19] "s---s" "t---t" "u---u" "v---v" "w---w" "x---x" "y---y" "z---z"
```

```
paste(letters, collapse = " ")
```

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

```
paste(letters,letters, sep = "---", collapse = " ")
```

```
[1] "a---a b---b c---c d---d e---e f---f g---g h---h i---i j---j k---k l---l m---m n---n o---
```

## Nettoyage de l'environnement de travail

Avant de passer au défi, nettoyons l'environnement de travail. C'est une chose que nous devons souvent faire pour éviter l'accumulation d'objets inutiles et l'encombrement de la mémoire. Le nettoyage peut être complet ou sélectif. Voyez ci-dessous:

```
rm(list=setdiff(ls(), "citation_proust_1")) # Élimination de tous les objets, à l'exception de citation_proust_1

rm(list=ls()) # Élimination de tous les objets de l'environnement

gc() # Permet de libérer l'espace mémoire non utilisée
```

	used (Mb)	gc trigger	(Mb) limit	(Mb) max used	(Mb)
Ncells	952383	50.9	2012831	107.5	NA
Vcells	3170933	24.2	8388608	64.0	65536 7891002 60.3

## Liste des fonctions utilisées dans cet atelier

Table 1: Fonctions utilisées dans cet atelier

Nom de la fonction	Extension d'origine	Sortie
<code>list.files()</code>	base	Vecteur contenant les noms des éléments d'un dossier
<code>install.packages()</code>	base	Installation d'une extension à partir de CRAN
<code>library()</code>	base	Activation d'une extension installée
<code>help()</code>	base	Aide sur une fonction ou une extension
<code>proust_random()</code>	proustr	Une phrase tirée de <i>La Recherche du temps perdu</i>
<code>nchar()</code>	base	Nombre de caractères d'une chaîne
<code>length()</code>	base	Nombre d'éléments d'un objet. Appliquée à une table, la fonction renverra le nombre de colonnes.
<code>mode()</code>	base	Type de structure d'un objet
<code>print()</code>	base	Contenu d'un objet
<code>str()</code>	base	Structure d'un objet
<code>identical()</code>	base	Identité de deux objets. Renvoie TRUE si les objets sont identiques et FALSE s'il y a la moindre différence entre les deux objets.
<code>paste()</code>	base	Un vecteur de longueur 1 si l'argument <code>collapse=</code> est utilisé, ou d'une longueur équivalente au nombre d'éléments joints.

## Défi

1. Assignez à cinq objets différents une citation aléatoire de Proust composée chacune d'une seule phrase;
2. Créez un objet de longueur 5 avec ces cinq vecteurs atomiques;
3. Amalgamez (collez) le premier et le dernier élément de ce vecteur, en les séparant avec une série d'astérisques `*****`.
4. Faites une copie du vecteur `letters`.
5. Utilisez l'indigage et la fonction `paste()` pour former le mot "abc".

## **Pour aller plus loin**

Colin Fay, [ProustR](#)

Vincent Goulet, David Beauchemin et Samuel Cabral Cruz [Introduction à R](#)

Sophie Baillargeon, [Présentation de R](#)

[Stack Overflow](#)