# Notes de leçon | Les bases du codage

## GRAPH Network & OMS, soutenu par le Fonds Mondial

### January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.

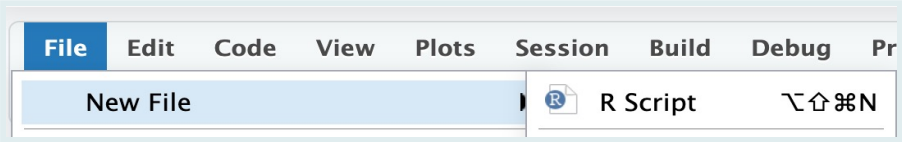
## Objectifs d'apprentissage

1. Vous pouvez écrire des commentaires dans R.
2. Vous pouvez créer des en-têtes de section dans RStudio.
3. Vous savez utiliser R comme calculatrice.
4. Vous pouvez créer, écraser et manipuler des objets R.
5. Vous comprenez les règles de base pour nommer les objets R.
6. Vous comprenez la syntaxe pour appeler les fonctions R.
7. Vous savez comment imbriquer plusieurs fonctions.
8. Vous pouvez utiliser, installer et charger des compléments de packages R et appeler des fonctions à partir de ces packages.

## Introduction

Dans la dernière leçon, vous avez appris à utiliser RStudio, le merveilleux environnement de développement intégré (IDE) qui facilite grandement le travail avec R. Dans cette leçon, vous apprendrez les bases de l'utilisation de R lui-même.

Pour commencer, ouvrez RStudio et ouvrez un nouveau script avec `File > New File > R Script` dans le menu RStudio.




Ensuite, **enregistrez le script** avec `File > Save` dans le menu de RStudio ou en utilisant le raccourci `Commande/Contrôle + S` . Cela devrait faire apparaître la boîte de dialogue "Save file". Enregistrez le fichier avec un nom comme "coding_basics".

Vous devez maintenant saisir tout le code de cette leçon dans ce script.

## Commentaires

Il existe deux principaux types de texte dans un script R : les commandes et les commentaires. Une commande est une ou plusieurs lignes de code R qui ordonne à R de faire quelque chose (par exemple `2 + 2`).

Un commentaire est un texte qui est ignoré par l'ordinateur.

Tout ce qui suit le symbole # (prononcé "dièse") sur une ligne donnée est un commentaire. Essayez de taper et d'exécuter le code ci-dessous pour voir ceci :

```
# Un commentaire
2 + 2 # Un autre commentaire
# 2 + 2
```

Puisqu'ils sont ignorés par l'ordinateur, les commentaires sont destinés aux *humains*. Ils vous aident, vous et les autres, à garder une trace de ce que fait votre code. Utilisez-les souvent ! Comme votre mère le dit toujours, "Tout excès nuit, sauf les commentaires R".

**Question 1**

Vrai ou Faux : les deux morceaux de code ci-dessous sont des façons valables de commenter du code ?

```
# Additionnez deux nombres
2 + 2
```

```
2 + 2 # Additionnez deux nombres
```

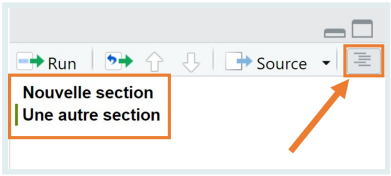
**Remarque:** Toutes les réponses aux questions se trouvent à la fin de la leçon.

Une utilisation fantastique des commentaires consiste à séparer vos scripts en sections. Si vous mettez quatre tirets après un commentaire, RStudio créera une nouvelle section dans votre code :

```
# Nouvelle section ----
```

Cela présente deux avantages intéressants. Tout d'abord, vous pouvez cliquer sur la petite flèche située à côté de l'en-tête de la section pour replier ou réduire cette section de code :


Deuxièmement, vous pouvez cliquer sur l'icône « Outline » en haut à droite de l'Éditeur pour afficher et parcourir tout le contenu de votre script :




## R comme une calculatrice

R fonctionne comme une calculatrice et respecte l'ordre correct des opérations. Saisissez et exécutez les expressions suivantes et observez leur résultat :

```
2 + 2
```

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

```
2 - 2
```

```
## [1] 0
```

```
2 * 2 # deux fois deux
```

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

```
2 / 2 # deux divisé par deux
```

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

```
2 ^ 2 # deux élevé à la puissance deux
```

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

```
2 + 2 * 2  # ceci est évalué suivant l'ordre des opérations
```

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

```
sqrt(100)  # racine carrée
```

```
## [1] 10
```

La commande de racine carrée affichée sur la dernière ligne est un bon exemple de *fonction* R, où `100` est l'*argument* de la fonction. Vous verrez bientôt d'autres fonctions.


Nous espérons que vous vous souvenez du raccourci pour exécuter le code !

Pour **exécuter une seule ligne de code**, placez votre curseur n'importe où sur cette ligne, puis appuyez sur `Commande` + `Entrée` sur macOS, ou `Contrôle` + `Entrée` sur Windows.

Pour **exécuter plusieurs lignes**, faites glisser votre curseur pour surligner les lignes concernées, puis appuyez à nouveau sur `Commande`/`Contrôle` + `Entrée`.

**Question 2**

Dans l'expression suivante, quel signe est évalué en premier par R, le moins ou la division ?

```
2 - 2 / 2
```

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

## Mise en forme du code

R ne se soucie pas de la façon dont vous choisissez d'espacer votre code.

Pour les opérations mathématiques que nous avons effectuées ci-dessus, tout ce qui suit serait un code valide :

```
2+2
```

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

```
2 + 2
```

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

```
2                 +                     2
```

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

De même, pour la fonction `sqrt()` utilisée ci-dessus, n'importe lequel de ces codes serait valide :

```
sqrt(100)
```

```
## [1] 10
```

```
sqrt(    100     )
```

```
## [1] 10
```

```r
# vous pouvez même espacer la commande sur plusieurs lignes
sqrt(
  100
    )
```

```
## [1] 10
```

Mais bien sûr, vous devriez essayer d'espacer votre code de manière raisonnable. Qu'entend-on exactement par "raisonnable" ? Il peut être difficile pour vous de le savoir pour le moment. Avec le temps, en lisant le code d'autres personnes, vous apprendrez qu'il existe certaines *conventions* R pour l'espacement et la mise en forme du code.

En attendant, vous pouvez demander à RStudio de vous aider à formater votre code. Pour ce faire, mettez en évidence une section de code que vous voulez reformater, et, dans le menu de RStudio, allez à `Code > Reformat Code`, ou utilisez le raccourci `Maj + Commande/Contrôle + A`.

En attendant, vous pouvez demander à RStudio de vous aider à structurer votre code. Pour ce faire, mettez en surbrillance n'importe quelle section de code que vous souhaitez restructurer et, dans le menu de RStudio, accédez à `Code > Reformat Code`, ou utilisez le raccourci `Maj + Commande/Contrôlee + A`.


**Coincé sur le signe +**

Si vous exécutez une ligne de code incomplète, R affichera un signe "+" pour indiquer qu'il attend que vous terminiez le code.

Par exemple, si vous exécutez le code suivant :

```r
sqrt(100
```

vous n'obtiendrez pas la sortie que vous attendez (10). La console affichera plutôt `sqrt(` et un signe + :

```
> sqrt(100
+
```

R attend que vous complétiez la parenthèse fermante. Vous pouvez compléter le code et vous débarrasser du "+" en saisissant simplement la parenthèse manquante :

```r
)
```


```
> sqrt(100
+ )
[1] 10
```

Vous pouvez également appuyer sur la touche d'échappement "ESC" pendant que votre curseur est dans la console pour recommencer.

## Objets dans R

### Créer un objet

Lorsque vous exécutez du code comme nous l'avons fait ci-dessus, le résultat de la commande (ou sa *valeur*) est simplement affiché dans la console – il n'est stocké nulle part.

```
2 + 2 # R imprime ce résultat, 4, mais ne le stocke pas
```

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

Pour stocker une valeur pour une utilisation future, attribuez-la à un *objet* à l'aide de l'*opérateur d'attribution*, <– :

```
mon_obj <- 2 + 2 # attribue le résultat de `2 + 2 ` à l'objet appelé `mon_obj`
mon_obj # imprime mon_obj
```

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

L'opérateur d'attribution, <– , est composé du signe 'inférieur à', < , et d'un signe moins, –. Vous l'utiliserez des milliers de fois au cours de votre vie R, alors ne le saisissez pas manuellement ! Utilisez plutôt le raccourci de RStudio, **`alt`** + **–** (**alt** ET **moins**) sous Windows ou **`option`** + **–** (**option** ET **moins**) sur macOS.

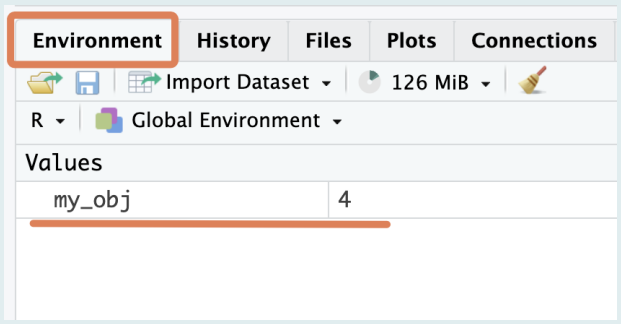

Notez également que vous pouvez utiliser le signe *égal*, =, pour l'attribution.

```
mon_obj = 2 + 2
```

**SIDE NOTE**

également. Suivez la convention et utilisez <-.

Maintenant que vous avez créé l'objet `mon_obj`, R sait tout à son sujet et en gardera la trace pendant cette session R. Vous pouvez voir tous les objets créés dans l'onglet *Environment* de RStudio.




## Qu'est-ce qu'un objet ?

Alors qu'est-ce qu'un objet exactement ? Considérez-le comme un seau nommé qui peut contenir n'importe quoi. Lorsque vous exécutez le code ci-dessous :

```
mon_obj <- 20
```

vous dites à R, "mettez le nombre 20 dans un seau nommé 'mon_obj'".

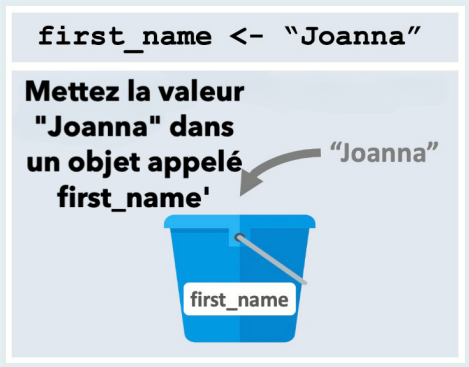

Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet appelé `mon_obj` est 20".

---

Et si vous exécutez ce code :

```
prenom <- "Joanna"
```

vous demandez à R de "mettre la valeur 'Joanna' dans le seau appelé `prenom`.




Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet `prenom` est Joanna".

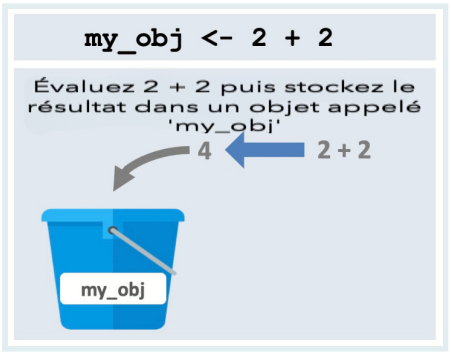
---

Notez que R évalue le code * avant * de le placer dans le seau.

Donc, avant l'exécution de ce code,

```
mon_obj <- 2 + 2
```

R effectue d'abord le calcul de `2 + 2`, puis stocke le résultat, 4, à l'intérieur de l'objet.




**Question 3**

Considérez le morceau de code ci-dessous :

```
resultat <- 2 + 2 + 2
```

Quelle est la valeur de l'objet `resultat` créé ?

A. `2 + 2 + 2`

B. numérique

C. 6

### Les jeux de données sont aussi des objets

Jusqu'à présent, vous avez travaillé avec des objets très simples. Vous pensez peut-être "Où sont les feuilles de calcul et les jeux de données ? Pourquoi écrivons-nous `mon_obj <- 2 + 2` ? Est-ce un cours de mathématiques à l'école primaire ?!"

Soyez patient.

Voyons maintenant un aperçu de cela. Saisissez le code ci-dessous pour télécharger un jeu de données sur les cas d'Ebola que nous avons stocké sur Google Drive et placez-le dans l'objet `ebola_sierra_leone_data`.

```
ebola_sierra_leone_data <- read.csv("https://tinyurl.com/ebola-data-sample")
ebola_sierra_leone_data # Affiche ebola_data
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
## 4 187  NA   F confirmed    2014-06-19     2014-06-24 Kailahun
## 5  85  20   M confirmed    2014-06-08     2014-06-24 Kailahun
```

Ces données contiennent un échantillon d'informations sur les patients de l'épidémie d'Ebola de 2014 à 2016 en Sierra Leone.

---

Étant donné que vous pouvez stocker des jeux de données en tant qu'objets, il est très facile de travailler avec plusieurs jeux de données en même temps.

Ci-dessous, nous importons et visualisons un autre jeu de données à partir du Web :

```
diabete_chine <- read.csv("https://tinyurl.com/diabetes-china")
```

Étant donné que le jeu de données ci-dessus est assez volumineux, il peut être utile de le consulter dans le visualiseur de données :

```
View(diabete_chine)
```

Notez que les deux jeux de données apparaissent maintenant dans votre onglet *Environment*.


Plutôt que de lire les données à partir d'un lecteur Internet comme nous l'avons fait ci-dessus, il est plus probable que vous ayez les données sur votre ordinateur et que vous souhaitiez les lire dans R à partir de là. Nous aborderons ce point dans une prochaine leçon.


Plus tard dans le cours, nous vous montrerons également comment stocker et lire des données à partir d'un service Web comme Google Drive, ce qui est pratique pour faciliter la portabilité.

### Renommer un objet

Vous souhaitez parfois renommer un objet. Il n'est pas possible de le faire directement.

Pour renommer un objet, faites une copie de l'objet avec un nouveau nom et supprimez l'original.

Par exemple, nous pouvons décider que le nom de l'objet `ebola_sierra_leone_data` est trop long. Pour le remplacer par le nom plus court "ebola_data" et exécuter :

```
ebola_data <- ebola_sierra_leone_data
```

Cela a copié le contenu du *seau* `ebola_sierra_leone_data` vers un nouveau *seau* `ebola_data`.

Vous pouvez maintenant vous débarrasser de l'ancien seau `ebola_sierra_leone_data` avec la fonction `rm()`, qui signifie "supprimer":

```
rm(ebola_sierra_leone_data)
```

### Écraser un objet

Écraser un objet revient à changer le *contenu* d'un *seau*.

Par exemple, nous avons précédemment exécuté ce code pour stocker la valeur "Joanna" dans l'objet `prenom`:

```
prenom <- "Joanna"
```

Pour changer cette valeur, réexécutez simplement la ligne avec une valeur différente :

```
prenom <- "Luigi"
```

Vous pouvez jeter un coup d'œil à l'onglet Environment pour observer le changement.

### Travailler avec des objets

Vous passerez la plupart de votre temps dans R à manipuler des objets R. Voyons quelques exemples rapides.

Vous pouvez exécuter des commandes simples sur les objets. Par exemple, ci-dessous, nous stockons la valeur `100` dans un objet, puis nous prenons la racine carrée de l'objet :

Vous pouvez exécuter des commandes simples sur des objets. Par exemple, ci-dessous, nous stockons la valeur `100` dans un objet, puis nous prenons la racine carrée de l'objet :

```
mon_nombre <- 100
sqrt(mon_nombre)
```

```
## [1] 10
```

R "voit" `mon_nombre` comme le nombre 100, et est donc capable d'évaluer sa racine carrée.

---

Vous pouvez également combiner des objets existants pour en créer de nouveaux. Par exemple, saisissez le code ci-dessous pour ajouter `mon_nombre` à lui-même et stockez le résultat dans un nouvel objet appelé `ma_somme` :

```
ma_somme <- mon_nombre + mon_nombre
```

Quelle devrait être la valeur de `ma_somme` ? Essayez d'abord de deviner, puis vérifiez.


Pour vérifier la valeur d'un objet, tel que `ma_somme`, vous pouvez saisir et exécuter uniquement le code `ma_somme` dans la console ou l'éditeur. Alternativement, vous pouvez simplement mettre en surbrillance la valeur `ma_somme` dans le code existant et appuyer sur `Commande/Contrôle` + `Entrée`.

---

Mais bien sûr, la plupart de votre analyse impliquera de travailler avec des objets *données*, tels que l'objet `ebola_data` que nous avons créé précédemment.

Voyons un exemple très simple de la façon d'interagir avec un objet de données ; nous l'aborderons plus en détail dans la prochaine leçon.

Pour obtenir un tableau de la répartition des patients par sexe dans l'objet `ebola_data`, nous pouvons exécuter ce qui suit :

```
table(ebola_data$sex)
```

```
##
##   F   M
## 124  76
```

Le symbole du dollar, $, ci-dessus nous a permis de créer un sous-ensemble dans une colonne spécifique.

**Question 4**

a. Considérez le code ci-dessous. Quelle est la valeur de l'objet `reponse` ?

```
huit <- 9
reponse <- huit - 8
```

b. Utilisez `table()` pour créer un tableau montrant la répartition des patients dans les districts dans l'objet `ebola_data`.

### Quelques erreurs avec les objets

```
prenom <- "Luigi"
nom <- "Fenway"
```

```
nom_complet <- prenom + nom
```

```
Erreur dans prenom + nom : argument non numérique à l'opérateur binaire
```

Le message d'erreur vous indique que ces objets ne sont pas des nombres et ne peuvent donc pas être additionnés avec +. Il s'agit d'un type d'erreur assez courant, causé par la tentative de faire des choses inappropriées à vos objets. Soyez prudent à ce sujet.

Dans ce cas particulier, nous pouvons utiliser la fonction `paste()` pour assembler ces deux objets :

```
nom_complet <- paste(prenom, nom)
nom_complet
```

```
## [1] "Luigi Fenway"
```

---

Une autre erreur que vous obtiendrez souvent est `Erreur : l'objet 'XXX' est introuvable`. Par example:

```
mon_nombre <- 48 # définit `mon_nombre`
Mon_nombre + 2 # essaie d'ajouter 2 à `Mon_nombre`
```

```
Erreur : objet 'Mon_nombre' introuvable
```

Ici, R renvoie un message d'erreur car nous n'avons pas encore créé (ou *défini*) l'objet `Mon_nombre`. (Rappelez-vous que R est sensible aux majuscules et minuscules.)

---

Lorsque vous commencez à apprendre R, gérer les erreurs peut être frustrant. Elles sont souvent difficiles à comprendre (par exemple, que signifie exactement "*argument non numérique à l'opérateur binaire*" ?).

Essayez de chercher sur Google tous les messages d'erreur que vous obtenez et parcourez les premiers résultats. Cela vous mènera à des forums (par exemple stackoverflow.com) où d'autres apprenants de R se sont plaints de la même erreur. Vous y trouverez des explications et des solutions à vos problèmes.

**Question 5**

a. Le code ci-dessous renvoie une erreur. Pourquoi?

```
mon_prenom <- "Kene"
mon_nom <- "Nwosu"
mon_prenom + mon_nom
```

b. Le code ci-dessous renvoie une erreur. Pourquoi? (Regardez attentivement)

```
mon_1er_prenom <- "Kene"
mon_nom <- "Nwosu"

paste(mon_Ier_prenom, mon_nom)
```

## Nommer les objets

> Il n'y a que ***deux choses difficiles*** en informatique : l'invalidation du cache et ***nommer les objets***.
>
> – Phil Karlton.

Étant donné qu'une grande partie de votre travail dans R implique d'interagir avec des objets que vous avez créés, il est important de choisir des noms intelligents pour ces objets.

Nommer des objets est difficile car les noms doivent être à la fois **courts** (afin que vous puissiez les taper rapidement) et **informatifs** (afin que vous puissiez facilement vous souvenir de ce qu'il y a à l'intérieur de l'objet), et ces deux objectifs sont souvent en conflit.

Ainsi, les noms trop longs, comme celui ci-dessous, sont mauvais parce qu'ils sont longs à saisir.

```
exemple_de_donnees_sur_lepidemie_debola_en_sierra_leone_en_2014
```

Et un nom comme `données` est mauvais car il n'est pas informatif ; le nom ne donne pas une bonne idée de ce qu'est l'objet.

Au fur et à mesure que vous écrivez du code R, vous apprendrez à écrire des noms courts et informatifs.

---

Pour les noms composés de plusieurs mots, il existe quelques conventions pour séparer les mots :

```
forme_serpent <- "La forme serpent utilise les traits de soulignement"
forme.pointe <- "La forme pointe utilise des points"
formeChameau <- "La forme chameau met en majuscule les nouveaux mots (Mais pas le premier mot)"
```

Nous recommandons la forme_serpent, qui utilise tous les mots en minuscules et sépare les mots par _.

---

Notez également qu'il existe certaines restrictions sur les noms d'objets :

- les noms doivent commencer par une lettre. Ainsi, `2014_data` n'est pas un nom valide (car il commence par un chiffre).
- les noms ne peuvent contenir que des lettres, des chiffres, des points (`.`) et des traits de soulignement (_). Donc `ebola-data` ou `ebola~data` ou `ebola data` avec un espace ne sont pas des noms valides.

Si vous voulez vraiment utiliser ces caractères dans vos noms d'objets, vous pouvez entourer les noms avec des accents graves :

```
`ebola-données`
`ebola~données`
`données Ebola`
```

Tous les noms ci-dessus sont des noms d'objet R valides. Par exemple, saisissez et exécutez le code suivant :

```
`ebola~data` <- ebola_data
`ebola~data`
```

Mais en général, vous devriez éviter d'utiliser des accents graves pour remédier aux mauvais noms d'objets. Écrivez simplement les noms propres.

**Question 6**

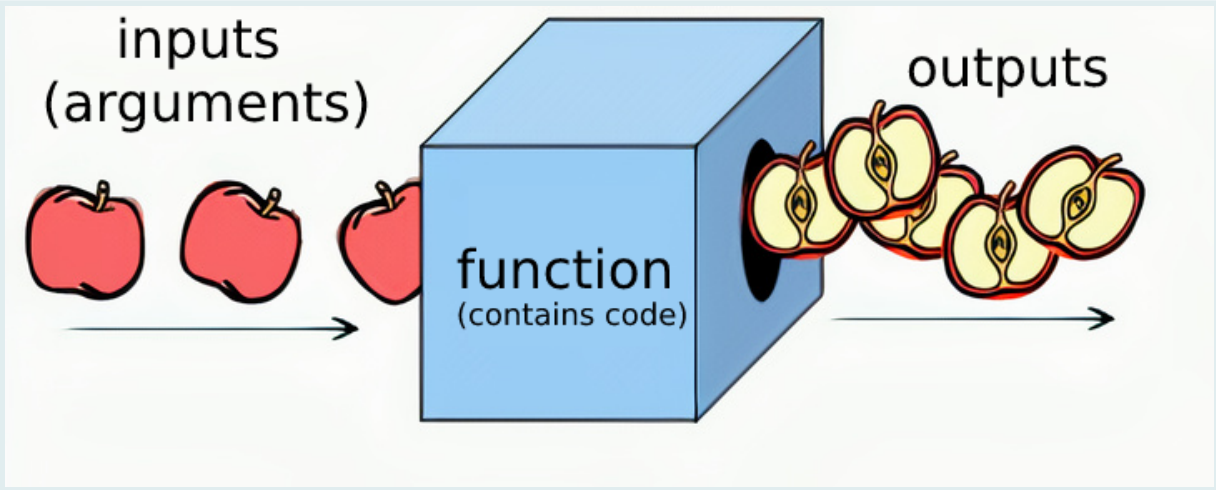
Dans le morceau de code ci-dessous, nous essayons de prendre les 20 premières lignes du tableau `ebola_data`. Toutes ces lignes sauf une comportent une erreur. Quelle ligne fonctionnera correctement ?

```r
20_premieres_lignes <- head(ebola_data, 20)
vingt-premieres-lignes <- head(ebola_data, 20)
premiere_20_lignes <- head(ebola_data, 20)
```

## Les fonctions

Une grande partie de votre travail dans R consistera à appeler des *fonctions*.

Vous pouvez considérer chaque fonction comme une machine qui reçoit certaines entrées (ou *arguments*) et renvoie des données de sortie.




Jusqu'à présent, vous avez déjà vu de nombreuses fonctions, notamment `sqrt()`, `paste()` et `plot()`. Exécutez les lignes ci-dessous pour vous rafraîchir la mémoire :

```r
sqrt(100) # Racine carrée
paste("Je suis le nombre", 2 + 2) # Coller
plot(women) # Fait ressortir le graphique
```

### Syntaxe de la fonction de base

La façon standard d'appeler une fonction est de fournir une *valeur* pour chaque *argument* :

```r
nom_de_la_fonction(argument1 = "valeur", argument2 = "valeur")
```

Démontrons ceci avec la fonction `head()`, qui renvoie les premiers éléments d'un objet.

Pour retourner les trois premières lignes de le jeu de données Ebola, exécutez :

```r
head(x = ebola_data, n = 3)
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
```

Dans le code ci-dessus, `head()` prenons deux arguments :

- `x` , l'objet qui nous intéresse, et
- `n`, le nombre d'éléments à retourner.

On peut aussi permuter l'ordre des arguments :

```
head(n = 3, x = ebola_data)
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
```

---

Si vous placez les valeurs des arguments dans le bon ordre, vous pouvez éviter de saisir de leurs noms. Ainsi, les deux lignes de code suivantes sont équivalentes et s'exécutent toutes les deux :

```
head(x = ebola_data, n = 3)
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
```

```
head(ebola_data, 3)
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
```

Mais si les valeurs des arguments sont dans le mauvais ordre, vous obtiendrez une erreur si vous ne saisissez pas les noms des arguments. Ci-dessous, la première ligne fonctionne mais la seconde ne fonctionne pas :

```
head(n = 3, x = ebola_data)
head(3, ebola_data)
```

(Pour connaître "l'ordre correct" des arguments, consultez le fichier d'aide de la fonction `head()`)

---

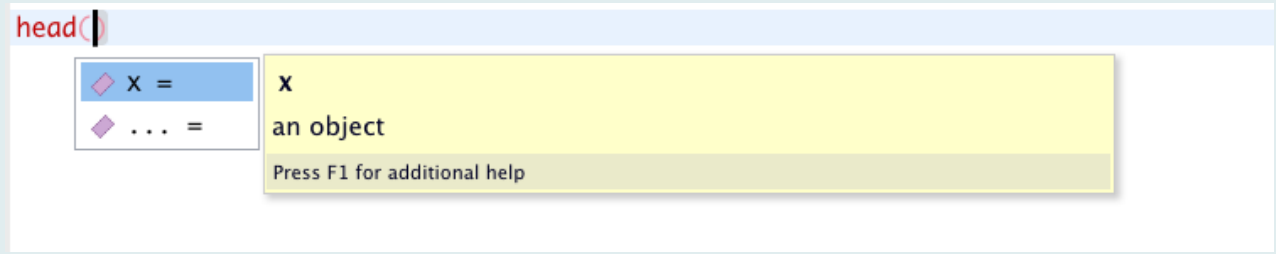
Certains arguments de fonction peuvent être ignorés, car ils ont des valeurs *par défaut*.

Par exemple, avec `head()`, la valeur par défaut de `n` est 6, donc exécuter uniquement `head(ebola_data)` renverra les 6 premières lignes.

```
head(ebola_data)
```

```
##    id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15     2014-06-21   Kenema
## 2 129  41   M confirmed    2014-06-13     2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28     2014-07-03 Kailahun
## 4 187  NA   F confirmed    2014-06-19     2014-06-24 Kailahun
## 5  85  20   M confirmed    2014-06-08     2014-06-24 Kailahun
## 6 277  30   F confirmed    2014-06-29     2014-07-01   Kenema
```

Pour voir les arguments d'une fonction, appuyez sur la touche **Tab** lorsque votre curseur se trouve à l'intérieur des parenthèses de la fonction :




**Question 7**

Dans les lignes de code ci-dessous, nous essayons de prendre les 6 premières lignes de le jeu de données `women` (qui est intégré à R). Quelle ligne n'est pas valide ?

```
head(women)
head(women, 6)
head(x = women, 6)
head(x = women, n = 6)
head(6, women)
```

(Si vous n'êtes pas sûr, essayez simplement de saisir et d'exécuter chaque ligne. N'oubliez pas que le but ici est que vous faire acquérir un peu de pratique.)

---

n'importe quel nombre d'arguments d'entrée.

Vous pouvez donc avoir soit deux arguments :

```
paste("Luigi", "Fenway")
```

```
## [1] "Luigi Fenway"
```

Soit quatre arguments :

```
paste("Luigi", "Fenway", "Luigi", "Fenway")
```

```
## [1] "Luigi Fenway Luigi Fenway"
```

Et ainsi de suite jusqu'à l'infini.

Et comme vous vous en souvenez peut-être, nous pouvons également coller (`paste()`) des objets nommés :

```
prenom <- "Luigi"
paste("Mon prenom est", prenom, "et mon nom est", nom)
```

```
## [1] "Mon prenom est Luigi et mon nom est Fenway"
```


Les fonctions comme `paste()` peuvent prendre plusieurs valeurs car elles ont un argument spécial, des points de suspension : **...** Si vous consultez le fichier d'aide de la fonction paste, vous verrez ceci :

**Arguments**

| | |
|---|---|
| `...` | one or more R objects, to be converted to character vectors. |

Un autre argument utile pour `paste()` est appelé `sep`. Il indique à R quel caractère utiliser pour séparer les termes :

```
paste("Luigi", "Fenway", sep = "-")
```

```
## [1] "Luigi-Fenway"
```

### L' imbrication des fonctions

La résultante d'une fonction peut être immédiatement prise en compte par une autre fonction. C'est ce qu'on appelle l'imbrication de fonctions.

Par exemple, la fonction `tolower()` convertit une chaîne de caractères en minuscules.

```
tolower("LUIGI")
```

```
## [1] "luigi"
```

Vous pouvez prendre la sortie de cette fonction et la passer directement dans une autre fonction :

```
paste(tolower("LUIGI"), "est mon nom")
```

```
## [1] "luigi est mon nom"
```

---

Sans cette option d'imbrication, il faudrait affecter un objet intermédiaire :

```
mon_nom_minuscule <- tolower("LUIGI")
paste(mon_nom_minuscule, "est mon nom")
```

```
## [1] "luigi est mon nom"
```

L'imbrication des fonctions sera bientôt très utile.

**Question 8**

Les morceaux de code ci-dessous sont tous des exemples d'imbrication de fonctions. Une des lignes contient une erreur. De quelle ligne s'agit-il et quelle est l'erreur ?

```
sqrt(head(women))
```

```
paste(sqrt(9), "plus 1 est", sqrt(16))
```

```
sqrt(tolower("LUIGI"))
```

## Les packages

Comme nous l'avons mentionné précédemment, R est formidable car il est extensible par l'utilisateur : n'importe qui peut créer un *package* logiciel qui ajoute de nouvelles fonctionnalités. La majeure partie de la puissance de R provient de ces packages.

Dans la leçon précédente, vous avez installé et chargé le package {highcharter} à l'aide des fonctions `install.packages()` et `library()`. Apprenons-en un peu plus sur les packages maintenant.

### Un premier exemple : le package {tableone}

Installons et utilisons maintenant un autre package R, appelé `tableone` :

```
install.packages("tableone")
```

```
library(tableone)
```

Notez que vous n'avez besoin d'installer un package qu'une seule fois, mais vous devez le charger avec `library()` chaque fois que vous voulez l'utiliser. Cela signifie que vous devez généralement exécuter la ligne `install.packages()` directement depuis la console, plutôt que de la saisir dans votre script.

---

Le package facilite la construction du "Tableau 1", c'est-à-dire un tableau avec les caractéristiques de l'échantillon d'étude que l'on trouve couramment dans les articles de recherche biomédicale.

Le cas d'utilisation le plus simple consiste à résumer le jeu de données. Il suffit d'introduire la base de données dans l'argument "data" de la fonction principale "CreateTableOne()".

```
CreateTableOne(data = ebola_data)
```

```
##
##                          Overall
##   n                         200
##   id (mean (SD))         146.00 (82.28)
##   age (mean (SD))         33.12 (17.85)
##   sex = M (%)                76 (38.0)
##   status = suspected (%)     18 ( 9.0)
##   date_of_onset (%)
##      2014-05-18               1 ( 0.5)
##      2014-05-20               1 ( 0.5)
##      2014-05-21               1 ( 0.5)
##      2014-05-22               2 ( 1.0)
##      2014-05-23               1 ( 0.5)
##      2014-05-24               2 ( 1.0)
```

```
##      2014-05-26               8 ( 4.0)
##      2014-05-27               7 ( 3.5)
##      2014-05-28               1 ( 0.5)
##      2014-05-29               9 ( 4.5)
##      2014-05-30               4 ( 2.0)
##      2014-05-31               2 ( 1.0)
##      2014-06-01               2 ( 1.0)
##      2014-06-02               1 ( 0.5)
##      2014-06-03               1 ( 0.5)
##      2014-06-05               1 ( 0.5)
##      2014-06-06               5 ( 2.5)
##      2014-06-07               3 ( 1.5)
##      2014-06-08               4 ( 2.0)
##      2014-06-09               1 ( 0.5)
##      2014-06-10              22 (11.0)
##      2014-06-11               1 ( 0.5)
##      2014-06-12               7 ( 3.5)
##      2014-06-13              15 ( 7.5)
##      2014-06-14               8 ( 4.0)
##      2014-06-15               3 ( 1.5)
##      2014-06-16               1 ( 0.5)
##      2014-06-17               4 ( 2.0)
##      2014-06-18               5 ( 2.5)
##      2014-06-19               8 ( 4.0)
##      2014-06-20               7 ( 3.5)
##      2014-06-21               2 ( 1.0)
##      2014-06-22               1 ( 0.5)
##      2014-06-23               2 ( 1.0)
##      2014-06-24               8 ( 4.0)
##      2014-06-25               6 ( 3.0)
##      2014-06-26              10 ( 5.0)
##      2014-06-27               9 ( 4.5)
##      2014-06-28              17 ( 8.5)
##      2014-06-29               7 ( 3.5)
##   date_of_sample (%)
##      2014-05-23               1 ( 0.5)
##      2014-05-25               1 ( 0.5)
##      2014-05-26               1 ( 0.5)
##      2014-05-27               2 ( 1.0)
##      2014-05-28               1 ( 0.5)
##      2014-05-29               2 ( 1.0)
##      2014-05-31               9 ( 4.5)
##      2014-06-01               6 ( 3.0)
##      2014-06-02               1 ( 0.5)
##      2014-06-03               9 ( 4.5)
##      2014-06-04               4 ( 2.0)
##      2014-06-05               1 ( 0.5)
##      2014-06-06               2 ( 1.0)
##      2014-06-07               2 ( 1.0)
##      2014-06-10               2 ( 1.0)
##      2014-06-11               4 ( 2.0)
##      2014-06-12               3 ( 1.5)
##      2014-06-13               3 ( 1.5)
##      2014-06-14               1 ( 0.5)
##      2014-06-15              21 (10.5)
##      2014-06-16               1 ( 0.5)
```

```
##       2014-06-17              5 ( 2.5)
##       2014-06-18             13 ( 6.5)
##       2014-06-19              9 ( 4.5)
##       2014-06-21              8 ( 4.0)
##       2014-06-22              7 ( 3.5)
##       2014-06-23              6 ( 3.0)
##       2014-06-24              6 ( 3.0)
##       2014-06-25              3 ( 1.5)
##       2014-06-27              5 ( 2.5)
##       2014-06-28              2 ( 1.0)
##       2014-06-29              8 ( 4.0)
##       2014-06-30              6 ( 3.0)
##       2014-07-01              4 ( 2.0)
##       2014-07-02             16 ( 8.0)
##       2014-07-03             13 ( 6.5)
##       2014-07-04              2 ( 1.0)
##       2014-07-05              2 ( 1.0)
##       2014-07-06              1 ( 0.5)
##       2014-07-08              3 ( 1.5)
##       2014-07-12              1 ( 0.5)
##       2014-07-14              1 ( 0.5)
##       2014-07-17              1 ( 0.5)
##       2014-07-21              1 ( 0.5)
##    district (%)
##       Bo                      4 ( 2.0)
##       Kailahun              146 (73.0)
##       Kenema                 41 (20.5)
##       Kono                    2 ( 1.0)
##       Port Loko               2 ( 1.0)
##       Western Urban           5 ( 2.5)
```

Vous pouvez voir qu'il y a 200 patients dans ce jeu de données, l'âge moyen est de 33 ans et 38% de l'échantillon de l'échantillon est de sexe masculin, entre autres détails.

Très cool! (Un problème est que le package suppose que les variables de date sont nominales; à cause de cela, le tableau de sortie est beaucoup trop long!)

Le but de cette démonstration de {tableone} est de vous montrer qu'il y a beaucoup de puissance dans les packages R externes. C'est une grande force de travailler avec R, un langage open-source avec un écosystème dynamique de contributeurs. Des milliers de personnes travaillent actuellement sur des packages qui pourraient vous être utiles un jour.

Vous pouvez rechercher sur Google "Cool R Packages" et parcourir les réponses si vous souhaitez en savoir plus sur d'autres packages R.

**SIDE NOTE**

Vous avez peut-être remarqué que nous mettons les noms de packages entre accolades, par ex. {tableone}. Il s'agit simplement d'une convention de style entre les utilisateurs/enseignants de R. Les accolades ne *signifient* rien.

### Signifiants complets

Le *signifiant complet* d'une fonction inclut à la fois le nom du package et le nom de la fonction : `package::function()`.

Ainsi par exemple, au lieu d'écrire :

```
CreateTableOne(data = ebola_data)
```

Nous pourrions écrire cette fonction avec son signifiant complet, `package::function()` :

```
tableone::CreateTableOne(data = ebola_data)
```

Vous n'avez généralement pas besoin d'utiliser ces signifiants complets dans vos scripts. Mais dans certaines situations, cela peut s'avérer utile :

La raison la plus courante est que vous souhaitez indiquer très clairement de quel package provient une fonction.

Deuxièmement, vous souhaitez parfois éviter d'avoir à exécuter `library(package)` avant d'accéder aux fonctions d'un package. En d'autres termes, vous souhaitez utiliser une fonction d'un package sans d'abord charger ce package à partir de la bibliothèque. Dans ce cas, vous pouvez utiliser la syntaxe complète du signifiant.

Donc ce qui suit :

```
tableone::CreateTableOne(data = ebola_data)
```

est équivalent à:

```
library(tableone)
CreateTableOne(data = ebola_data)
```

**Question 9**

Considérez le code ci-dessous :

```
tableone::CreateTableOne(data = ebola_data)
```

Lequel des énoncés suivants est une interprétation correcte de ce que signifie ce code :

A. Le code applique la fonction `CreateTableOne` du package {tableone} sur l'objet `ebola_data`.

B. Le code applique l'argument `CreateTableOne` de la fonction {tableone} sur le package `ebola_data`.

C. Le code applique la fonction `CreateTableOne` du package {tableone} sur le package `ebola_data`.

### pacman::p_load()

Plutôt que d'utiliser deux fonctions distinctes, `install.packages()` puis `library()`, pour installer puis charger des packages, vous pouvez utiliser une seule fonction, `p_load()`, du package {pacman} pour installer automatiquement un package s'il n'est pas encore installé, *et* chargez le package. Nous encourageons cette approche dans la suite de ce cours.

Installez {pacman} maintenant en exécutant ceci dans votre console :

```
install.packages("pacman")
```

À partir de maintenant, lorsque vous découvrirez un nouveau package, vous pouvez simplement utiliser `pacman::p_load(nom_du_package)` pour installer et charger le package :

Essayez ceci maintenant pour le package "outbreaks", que nous utiliserons bientôt :

```
pacman::p_load(outbreaks)
```

---

Maintenant, nous avons un petit problème. La merveilleuse fonction `pacman::p_load()` installe et charge automatiquement les packages.

Mais ce serait bien d'avoir du code qui installe automatiquement le package {pacman} lui-même, s'il est manquant sur l'ordinateur d'un utilisateur.

Mais si vous mettez la ligne `install.packages()` dans un script, comme ceci :

```
install.packages("pacman")
pacman::p_load(here, rmarkdown)
```

vous perdrez beaucoup de temps. Parce qu'à chaque fois qu'un utilisateur ouvre et exécute un script, il va *réinstaller* {pacman}, ce qui peut prendre un certain temps. Au lieu de cela, nous avons besoin d'un code qui *vérifie d'abord si pacman n'est pas encore installé* et l'installe si ce n'est pas le cas.

Nous pouvons le faire avec le code suivant :

```
if(!require(pacman)) install.packages("pacman")
```

Vous n'avez pas besoin de le comprendre pour le moment, car il utilise une syntaxe que vous n'avez pas encore apprise. Notez simplement que dans les prochains chapitres, nous commencerons souvent un script avec un code comme celui-ci :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(here, rmarkdown)
```

La première ligne installe {pacman} s'il n'est pas encore installé. La deuxième ligne utilise la fonction `p_load()` de {pacman} pour charger les packages restants (et `pacman::p_load()` installe tous les packages qui ne sont pas encore installés).

Ouf! J'espère que votre tête est encore intacte.

**Question 10**

Au début d'un script R, nous aimerions installer et charger le package appelé {janitor}. Parmi les morceaux de code suivants, quels sont ceux que nous vous recommandons d'inclure dans votre script ?

A.

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(janitor)
```

B

```
install.packages("janitor")
library(janitor)
```

C

```
install.packages("janitor")
pacman::p_load(janitor)
```

## Conclusion

Avec vos nouvelles connaissances sur les objets R, les fonctions R et les packages d'où proviennent les fonctions, vous êtes prêt, croyez-le ou non, à effectuer une analyse de données de base dans R. Nous allons nous lancer tête baissée dans la prochaine leçon. Au plaisir de vous y retrouver !

### Réponses

1. Vrai.
2. Le signe de division est évalué en premier.
3. La réponse est C. Le code `2 + 2 + 2` est évalué avant d'être stocké dans l'objet.
4. a. La valeur est 1. Le code est évalué à '9-8'.

b. table(ebola_data$district)

5. a. Il n'est pas possible d'additionner deux chaînes de caractères. L'addition ne fonctionne que pour les nombres.

   b. `mon_1er_nom` est initialement saisi avec le chiffre 1, mais dans la commande `paste()`, il est saisi avec la lettre "l".

6. La troisième ligne est la seule ligne avec un nom d'objet valide : `premieres_20_lignes`

7. La dernière ligne, `head(6, women)`, n'est pas valide car les arguments sont dans le mauvais ordre et ils ne sont pas nommés.

8. Le troisième morceau de code a un problème. Il tente de trouver la racine carrée d'un caractère, ce qui est impossible.

9. La première ligne, A, est l'interprétation correcte.

10. Le premier morceau de code est la méthode recommandée pour installer et charger le package {janitor}

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :


### KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

### LAMECK AGASA

Statistician/Data Scientist

### OLIVIA KEISER

Head of division of Infectious Diseases and Mathematical Modelling, University of Geneva

### LAURE NGUEMO

Data Science Education Officer
Gets very excited at the mention of data, especially health related data

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :


- "File:Apple slicing function.png." *Wikimedia Commons, le référentiel multimédia gratuit*. 1er octobre 2021, 04:26 UTC. 20 mars 2022, 17:27 <https://commons.wikimedia.org/w/index.php?title=File:Apple_slicing_function.png&oldid=594767630>.
- "PsyteachR | Compétences en données pour une recherche reproductible." 2021. Github.io. 2021. https://psyteahr.github.io/reprores-v2/index.html.
- Douglas, Alex, Deon Roos, Francesca Mancini, Ana Couto et David Lusseau. 2022. "Une introduction à R." Intro2r.com. 27 janvier 2022. https://intro2r.com/.