Lesson notes | Coding basics (Translation 2, Hamza from Fiverr)

Created by the GRAPH Courses team

January 2023

This document serves as an accompaniment for a lesson found on https://thegraphcourses.org.

The GRAPH Courses is a project of the Global Research and Analyses for Public Health (GRAPH) Network, a non-profit headquartered at the University of Geneva Global Health Institute, and supported by the World Health Organization (WHO) and other partners

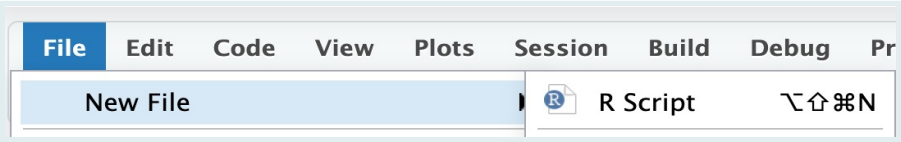
Objectifs d'apprentissage

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 imbriquer plusieurs fonctions.
8. Vous pouvez utiliser installer et charger des packages R complémentaires 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 `Command`/`Control` + `S` . Cela devrait faire apparaître la boîte de dialogue Enregistrer le fichier. 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 ligne ou des lignes de code R qui ordonne à R de faire quelque chose (par exemple `2 + 2`)

Un commentaire est un texte ignoré par l'ordinateur.

Tout ce qui suit un symbole `#` (prononcé "dièse" ou "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 suivre ce que fait votre code. Utilisez-les souvent ! Comme dit toujours ta mère, "trop tout va mal, sauf les commentaires R".

Question 1

Vrai ou faux : les deux morceaux de code ci-dessous sont des moyens valides de commenter le code : ?

```
# ajouter deux nombres
2 + 2
```

```
2 + 2 # additionne deux nombres
```

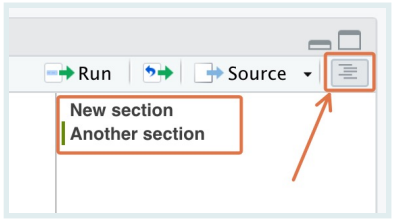
Remarque: Toutes les réponses aux questions se trouvent à la fin de la leçon.

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 rubrique ----
```

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

Deuxièmement, vous pouvez cliquer sur l'icône « Contour » 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 obéit au bon ordre des opérations. Tapez et exécutez les expressions suivantes et observez leur sortie :

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

Nous espérons que vous vous souviendrez 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" sous macOS ou sur "Contrôle" + "Entrée" sous Windows.

Pour **exécuter plusieurs lignes**, faites glisser votre curseur pour mettre en surbrillance les lignes pertinentes, puis appuyez à nouveau sur `Command`/`Control` + `Enter`.

Question 2

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

```
2 - 2 / 2
```

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

Code de formatage

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 éléments serait valide :

```
sqrt(100)
```

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

```
sqrt(    100      )
```

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

```
# 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 sensée. Qu'est-ce que le "sensé" exactement ? Eh bien, il peut être difficile pour vous de le savoir pour le moment. Au fil du temps, en lisant le code d'autres personnes, vous apprendrez qu'il existe certaines *conventions* R pour l'espacement et le formatage du code.

En attendant, vous pouvez demander à RStudio de vous aider à formater votre code. Pour ce faire, mettez en surbrillance n'importe quelle section de code que vous souhaitez reformater et, dans le menu RStudio, accédez à `Code> Reformater le code`, ou utilisez le raccourci `Shift` + `Command/Control` + `A`.

Coincé sur le signe +

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

Par exemple, si vous exécutez le code suivant :

```
sqrt(100
```

vous n'obtiendrez pas la sortie que vous attendez (10). Au lieu de cela, la console affichera `sqrt(` et un signe `+` :

```
> sqrt(100
+
```

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

```
)
```

```
> 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, affectez-la à un *objet* avec l'*opérateur d'affectation*, `<-` :

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

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

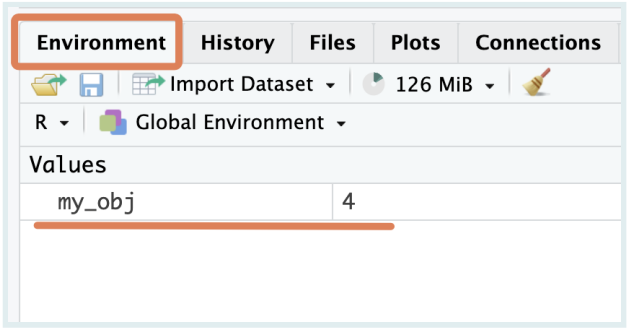
L'opérateur d'affectation, `<-` , est composé du signe 'inférieur à', `<` , et d'un signe moins, `-`. Vous l'utiliserez des milliers de fois au cours de votre vie R, alors ne le saisissez pas manuellement ! Utilisez plutôt le raccourci de RStudio, **`alt` + `-`** (**alt** ET **moins**) sous Windows ou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

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

```
my_obj = 2 + 2
```

Mais ce n'est pas couramment utilisé par la communauté R (principalement pour des raisons historiques), nous le déconseillons donc également. Suivez la convention et utilisez `<-`.

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

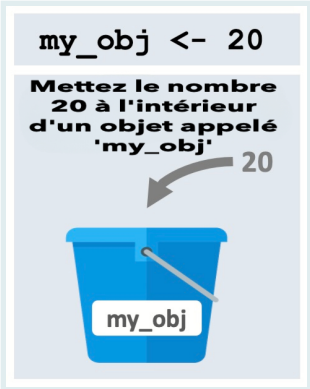


Qu'est-ce qu'un objet ?

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

```
my_obj <- 20
```

vous dites à R, "mettez le numéro 20 dans un seau nommé 'my_obj'".

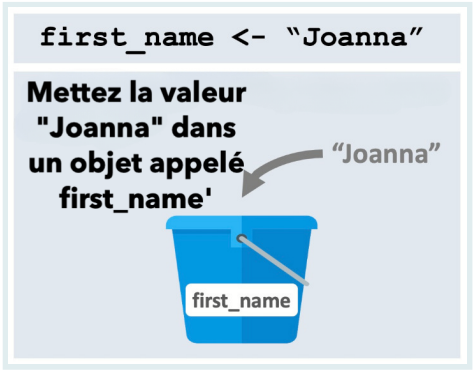


Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet appelé `my_obj` est 20".

Et si vous exécutez ce code :

```
first_name <- "Joanna"
```

vous demandez à R de "mettre la valeur 'Joanna' dans le compartiment appelé 'first_name'".



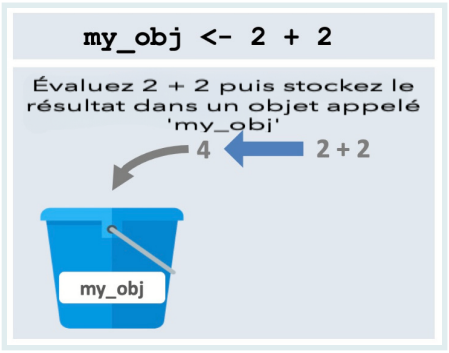
Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet `first_name` est Joanna".

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

Donc, avant quand nous avons exécuté ce code,

```
my_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 :

```
result <- 2 + 2 + 2
```

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

A. `2 + 2 + 2`

B. numérique

C. 6

Les ensembles 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 ensembles de données ? Pourquoi écrivons-nous `my_obj <- 2 + 2` ? Est-ce un cours de mathématiques à l'école primaire ?!"

Être patient.

Nous souhaitons que vous vous familiarisiez avec le concept d'objet R car une fois que vous aurez commencé à traiter des ensembles de données réels, ceux-ci seront également stockés en tant qu'objets R.

Voyons maintenant un aperçu de cela. Tapez le code ci-dessous pour télécharger un ensemble 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 # print 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 2014-2016 en Sierra Leone.

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

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

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

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

```
View(diabetes_china)
```

Notez que les deux ensembles de données apparaissent maintenant dans votre onglet *Environnement*.

::: side-note Plutôt que de lire les données 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 cela 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 agréable pour une portabilité facile. :::

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 décidons peut-être que le nom de l'objet `ebola_sierra_leone_data` est trop long. Pour le remplacer par l'exécution "ebola_data" plus courte :

```
ebola_data <- ebola_sierra_leone_data
```

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

Vous pouvez maintenant vous débarrasser de l'ancien bucket `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 `first_name` :

```
first_name <- "Joanna"
```

Pour changer cela en un autre, réexécutez simplement la ligne avec une valeur différente :

```
first_name <- "Luigi"
```

Vous pouvez jeter un œil à l'onglet Environnement pour observer le changement.

Travailler avec des objets

La plupart de votre temps dans R sera consacré à la manipulation d'objets R. Voyons quelques exemples rapides.

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

```
my_number <- 100
sqrt(my_number)
```

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

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

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

```
my_sum <- my_number + my_number
```

Quelle devrait être la valeur de `my_sum` ? Faites d'abord une supposition, puis vérifiez-la.

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

Mais bien sûr, la plupart de votre analyse impliquera de travailler avec des objets *data*, 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 correctement dans la prochaine leçon.

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

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

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

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

Question 4

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

```
eight <- 9
answer <- eight - 8
```

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

Quelques erreurs avec des objets

```
first_name <- "Luigi"
last_name <- "Fenway"
```

```
full_name <- first_name + last_name
```

```
Erreur dans first_name + last_name : argument non numérique vers un opérateur
binaire
```

Le message d'erreur vous indique que ces objets ne sont pas des nombres et ne peuvent donc pas être ajoutés 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 :

```
full_name <- paste(first_name, last_name)
full_name
```

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

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

```
my_number <- 48 # define `my_obj`
My_number + 2 # attempt to add 2 to `my_obj`
```

```
Erreur : objet 'Mon_numéro' introuvable
```

Ici, R renvoie un message d'erreur car nous n'avons pas encore créé (ou *défini*) l'objet `My_obj`. (Rappelez-vous que R est sensible à la casse.)

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

Essayez de googler 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 R se sont plaints de la même erreur. Vous trouverez ici des explications et des solutions à vos problèmes.

Question 5

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

```
my_first_name <- "Kene"
my_last_name <- "Nwosu"
my_first_name + my_last_name
```

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

```
my_1st_name <- "Kene"
my_last_name <- "Nwosu"

paste(my_Ist_name, my_last_name)
```

Nommer les objets

> Il n'y a que ***deux choses difficiles*** en informatique : l'invalidation du cache et ***nommer les choses***.
>
> – 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 car ils prennent une éternité à taper.

```
sample_of_the_ebola_outbreak_dataset_from_sierra_leone_in_2014
```

Et un nom comme `data` 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 :

```
snake_case <- "Snake case uses underscores"
period.case <- "Period case uses periods"
camelCase <- "Camel case capitalizes new words (but not the first word)"
```

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

Notez également qu'il existe certaines limitations 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 nombre).
- 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 de backticks :

```
`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 backticks pour sauver de mauvais noms d'objets. Écrivez simplement les noms propres.

Question 6

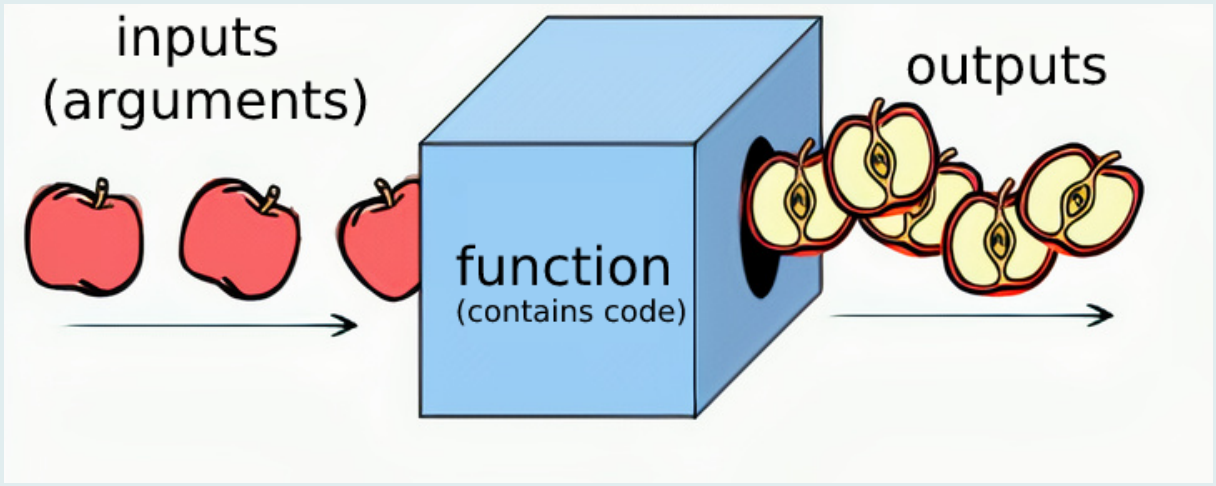
Dans le morceau de code ci-dessous, nous essayons de prendre les 20 premières lignes de la table `ebola_data`. Toutes ces lignes sauf une comportent une erreur. Quelle ligne fonctionnera correctement ?

```
20_top_rows <- head(ebola_data, 20)
twenty-top-rows <- head(ebola_data, 20)
top_20_rows <- head(ebola_data, 20)
```

Les fonctions

Une grande partie de votre travail dans R impliquera d'appeler des *fonctions*.

Vous pouvez considérer chaque fonction comme une machine qui accepte certaines entrées (ou *arguments*) et renvoie une 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 :

```
sqrt(100)
paste("I am number", 2 + 2)
plot(women)
```

Syntaxe de la fonction de base

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

```
function_name(argument1 = "value", argument2 = "value")
```

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

Pour renvoyer les trois premières lignes de l'ensemble de données Ebola, exécutez :

```
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()` prend deux arguments :

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

On peut aussi échanger 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 ignorer la saisie 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 tapez 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 voir "l'ordre correct" des arguments, consultez le fichier d'aide de la fonction `head()`)

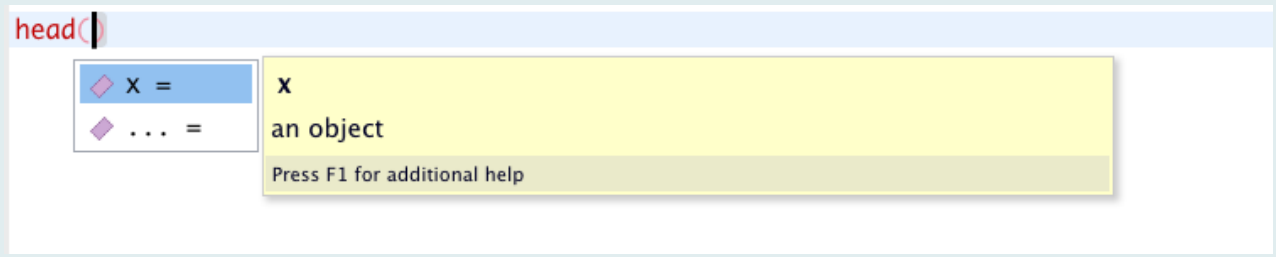
Certains arguments de fonction peuvent être complètement ignorés, car ils ont des valeurs *par défaut*.

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 l'ensemble de données "femmes" (qui est intégré à R). Quelle ligne est invalide ?

```
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 taper et d'exécuter chaque ligne. N'oubliez pas que le but ici est que vous acquériez un peu de pratique.)

Passons un peu de temps à jouer avec une autre fonction, la fonction `paste()`, que nous avons déjà vue ci-dessus. Cette fonction est un peu spéciale car elle peut prendre n'importe quel nombre d'arguments d'entrée.

Vous pourriez donc avoir 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()" des objets nommés :

```
first_name <- "Luigi"
paste("My name is", first_name, "and my last name is", last_name)
```

```
## [1] "My name is Luigi and my last name is 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"
```

Fonctions d'imbrication

La sortie 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 en minuscules.

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

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

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

```
paste(tolower("LUIGI"), "is my name")
```

```
## [1] "luigi is my name"
```

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

```
my_lowercase_name <- tolower("LUIGI")
paste(my_lowercase_name, "is my name")
```

```
## [1] "luigi is my name"
```

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 is", sqrt(16))
```

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

Paquets

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 paquet 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 l'ensemble de données. Vous pouvez simplement alimenter le bloc de données avec 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 cet ensemble de données, l'âge moyen est de 33 ans et 38% de l'échantillon de l'échantillon est un homme, entre autres détails.

Très cool! (Un problème est que le package suppose que les variables de date sont catégorielles ; à cause de cela, la table de sortie est beaucoup trop longue !)

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 "Packages Cool R" et parcourir les réponses si vous souhaitez en savoir plus sur les packages R.

Vous avez peut-être remarqué que nous incluons 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 il y a des situations où c'est 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. Autrement dit, 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

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

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

À partir de maintenant, lorsque vous découvrez un nouveau paquet, vous pouvez simplement utiliser `pacman::p_load(package_name)` pour installer et charger le paquet :

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 manque 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 *réinstalle* {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 installera {pacman} s'il n'est pas encore installé. La deuxième ligne utilisera 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).

Phew! J'espère que ta tête est encore intacte.

Question 10

Au début d'un script R, nous aimerions installer et charger le package appelé {janitor}. Lequel des morceaux de code suivants vous recommandons-nous d'avoir dans votre script ?

UN.

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

B

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

C

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

Emballer

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 aborderons cette question tête la première dans la prochaine leçon. On se voit là-bas!

Réponses {.unlisted .unnumbered}

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. Vous ne pouvez pas ajouter deux chaînes de caractères. L'addition ne fonctionne que pour les nombres.

   b. `my_1st_name` est initialement tapé avec le chiffre 1, mais dans la commande `paste()`, il est tapé avec la lettre "l".
6. La troisième ligne est la seule ligne avec un nom d'objet valide : `top_20_rows`

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

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/.