# Writing functions

# Viết hàm

## Preparation

## Chuẩn bị

### Load packages

### Tải gói lệnh

This code chunk shows the loading of packages required for the analyses. In this handbook we emphasize p\_load() from **pacman**, which installs the package if necessary *and* loads it for use. You can also load installed packages with library() from **base** R. See the page on [R basics](#bookmark=id.30j0zll) for more information on R packages.

Đoạn mã này hiển thị việc tải các gói lệnh cần thiết cho phân tích. Trong cuốn sổ tay này, chúng tôi nhấn mạnh đến hàm p\_load()trong gói lệnh **pacman**, cài đặt gói lệnh nếu cần thiết và tải để sử dụng. Các gói lệnh đã cài đặt cũng có thể được tải bằng library()từ **base** R. Xem thêm thông tin các gói lệnh của R trong chương [Cơ](#r-basics) bản về R.

### Import data

### Nhập số liệu

We import the dataset of cases from a simulated Ebola epidemic. If you want to download the data to follow step-by-step, see instructions in the [Download book and data] page. The dataset is imported using the import() function from the **rio** package. See the page on [Import and export](#bookmark=id.2et92p0) for various ways to import data.

We will also use in the last part of this page some data on H7N9 flu from 2013.

Chúng tôi nhập bộ số liệu của các ca bệnh được mô phỏng từ một vụ dịch Ebola. Nếu muốn tải số liệu để làm theo từng bước, xem hướng dẫn trong chương [Tải sách và số liệu](https://epirhandbook.com/download-handbook-and-data.html). Bộ số liệu được nhập vào bằng cách sử dụng hàm import()từ gói lệnh **rio**. Xem chương [Nhập và xuất số liệu](#bookmark=id.2et92p0) để biết các cách nhập dữ liệu khác.

Chúng tôi cũng sẽ sử dụng một vài số liệu về bệnh cúm do H7N9 xảy ra năm 2013 ở phần cuối của chương này.

## Functions

## Hàm

Functions are helpful in programming since they allow to make codes easier to understand, somehow shorter and less prone to errors (given there were no errors in the function itself).

If you have come so far to this handbook, it means you have came across endless functions since in R, every operation is a function call +, for, if, [, $, { …. For example x + y is the same as'+'(x, y)

Các hàm hữu ích trong lập trình vì chúng cho phép để tạo ra các mã dễ hiểu hơn, ngắn hơn và ít bị lỗi hơn (vì không có lỗi trong chính hàm này).

Nếu chúng ta đã xem đến chương này, có nghĩa là chúng ta đã xem qua chức năng vô tận, vì trong R, các mỗi thao tác là một lệnh gọi hàm +, for, if, [, $, { …. Ví dụ như x + y giống như '+'(x, y)

R is one the languages that offers the most possibility to work with functions and give enough tools to the user to easily write them. We should not think about functions as fixed at the top or at the end of the programming chain, R offers the possibility to use them as if they were vectors and even to use them inside other functions, lists…

Lot of very advanced resources on functional programming exist and we will only give here an insight to help you start with functional programming with short practical examples. You are then encouraged to visit the links on references to read more about it.

R là một ngôn ngữ cung cấp nhiều khả năng nhất để thực hiện các hàm và cung cấp đầy đủ các công cụ để người dùng dễ dàng viết chúng. Chúng ta không nên nghĩ các hàm là cố định ở đầu hay ở cuối chuỗi lập trình. R cung cấp các khả năng sử dụng chúng như thể chúng là các véc tơ và thậm chí sử dụng chúng bên trong các hàm, danh sách .v.v.

Có rất nhiều nguồn nâng cao về lập trình và chúng tôi chỉ cung cấp ở đây một cái nhìn sâu sắc để giúp bắt đầu lập trình với các ví dụ thực tế, ngắn gọn. Sau đó, chúng ta sẽ truy cập các tài liệu tham khảo để đọc thêm.

## Why would you use a function?

## Tại sao chúng ta sử dụng hàm?

Before answering this question, it is important to note that you have already had tips to get to write your very first R functions in the page on [Iteration, loops, and lists](#bookmark=id.4f1mdlm) of this handbook. In fact, use of “if/else” and loops is often a core part of many of our functions since they easily help to either broaden the application of our code allowing multiple conditions or to iterate codes for repeating tasks.

* I am repeating multiple times the same block of code to apply it to a different variable or data?
* Getting rid of it will it substantially shorten my overall code and make it run quicker?
* Is it possible that the code I have written is used again but with a different value at many places of the code?

If the answer to one of the previous questions is “YES”, then you probably need to write a function

Trước khi trả lời câu hỏi này, điều quan trọng cần lưu ý là chúng ta đã biết các mẹo để viết các hàm đầu tiên trong R trong chương [Lặp, vòng lặp và danh sách](#bookmark=id.4f1mdlm) trong cuốn sổ tay này. Nói tóm lại, việc sử dụng “if/else” và vòng lặp thường là một phần cốt lõi trong nhiều hàm vì chúng dễ dàng giúp mở rộng các ứng dụng mã mà cho phép nhiều điều kiện hoặc lặp lại mã cho các tác vụ lặp lại.

* Tôi có đang lặp lại nhiều lần cùng một khuôn mã để ứng dụng nó cho một biến hoặc số liệu khác không?
* Loại bỏ nó có rút ngắn đáng kể số mã tổng thể và giúp việc thực hiện nhanh hơn không?
* Đoạn mã có thể sử dụng lại với giá trị khác ở nhiều vị trí trong mã không?

Nếu một trong những trả lời cho các câu hỏi bên trên là “CÓ”, chắc hẳn rằng chúng ta cần phải viết hàm.

## How does R build functions?

## Cách viết hàm trong R

Functions in R have three main components:

* the formals() which is the list of arguments which controls how we can call the function
* the body() that is the code inside the function i.e. within the brackets or following the parenthesis depending on how we write it

and,

* the environment() which will help locate the function’s variables and determines how the function finds value.

Once you have created your function, you can verify each of these components by calling the function associated.

Các hàm trong R có ba thành phần chính:

* phần hình thức formals() là danh sách các đối số mà kiểm soát cách chúng ta có thể gọi hàm
* phần thân body() là mã bên trong hàm, tức là chức năng bên trong hoặc theo sau các dấu ngoặc mà tùy thuộc vào cách chúng ta viết nó

và,

* phần môi trường thực hiện environment() để giúp chúng ta xác định vị trí các biến trong hàm và xác định cách các hàm tìm giá trị.

Một khi đã tạo hàm, chúng ta cần kiểm tra mỗi thành phần này bằng cách gọi các hàm liên kết.

## Basic syntax and structure

## Cấu trúc và cú pháp cơ bản

* A function will need to be named properly so that its job is easily understandable as soon as we read its name. Actually this is already the case with majority of the base R architecture. Functions like mean(), print(), summary() have names that are very straightforward
* Một hàm cần được đặt tên theo đúng công việc của nó để có thể hiểu được khi chúng ta đọc tên hàm. Thực tế, điều này đã được sử dụng trong phần lớn các cấu trúc trong môi trường **base R.**  Các hàm như mean(), print(), summary() có những cái tên đúng như nhiệm vụ của nó.
* A function will need arguments, such as the data to work on and other objects that can be static values among other options
* Một hàm cần các đối số như là số liệu để thực hiện và các đối tượng khác mà có thể là các giá trị tĩnh trong số các tùy chọn khác.
* And finally a function will give an output based on its core task and the arguments it has been given. Usually we will use the built-in functions as print(), return()… to produce the output. The output can be a logical value, a number, a character, a data frame…in short any kind of R object.
* Và cuối cùng, một hàm sẽ cho kết quả đầu ra dựa trên nhiệm vụ cốt lõi của nó và các đối số đã được đưa ra. Thông thường, chúng ta sẽ sử dụng các hàm có sẵn như print(), return().v.v. để tạo ra kết quả đầu ra. Kết quả đầu ra có thể là giá trị logic, một số, một ký tự, một khung số liệu .v.v. mà gọi ngắn gọn là đối tượng của R.

Basically this is the composition of a function:

Về cơ bản, đây là thành phần của một hàm:

We can create our first function that will be called contain\_covid19().

Chúng ta tạo hàm đầu tiên với tên gọi là contain\_covid19().

We can then verify the components of our newly created function.

Sau đó, chúng ta kiểm tra các thành phần của hàm vừa mới được tạo ra.

Now we will test our function. To call our written function, you use it as you use all R functions i.e by writing the function name and adding the required arguments.

Bây giờ, chúng ta sẽ kiểm tra hàm này. Để gọi hàm đã biết, chúng ta sử dụng nó như sử dụng tất cả các hàm trong R, tức là bằng cách viết tên hàm và thêm các đối số cần phải có.

We can write again the name of each argument for precautionary reasons. But without specifying them, the code should work since R has in memory the positioning of each argument. So as long as you put the values of the arguments in the correct order, you can skip writing the arguments names when calling the functions.

Chúng ta có thể viết lại tên của mỗi đối số để đề phòng. Nhưng khi không cụ thể chúng, mã cũng sẽ thực hiện vì R đã nhớ vị trí của mỗi đối số. Vì thế, miễn là chúng ta đặt các giá trị của các đối số theo đúng thứ tự, chúng ta có thể bỏ qua việc viết tên các đối số khi gọi hàm.

Then let’s look what happens if one of the values is "no" or **not** "yes".

Sau đó, hãy xem điều gì sẽ xảy ra nếu một trong các giá trị là "no" hoặc **không** là "yes".

If we provide an argument that is not recognized, we get an error:

Error in contain\_covid19(barrier\_gest = "sometimes", wear\_mask = "yes", : could not find function "contain\_covid19"

Nếu chúng ta cung cấp các đối số không được công nhận, chúng ta sẽ gặp lỗi:

Lỗi trong hàm contain\_covid19(barrier\_gest = "sometimes", wear\_mask = "yes", : không tìm thầy hàm "contain\_covid19"

***NOTE:*** Some functions (most of time very short and straightforward) may not need a name and can be used directly on a line of code or inside another function to do quick task. They are called **anonymous functions** .

***LƯU Ý:*** Một số hàm (hầu hết là rất ngắn và không phức tạp) có thể không cần tên và có thể sử dụng trực tiếp trên một dòng mã hoặc bên trong một hàm khác để thực hiện tác vụ nhanh chóng. Các hàm này được gọi là **hàm ẩn danh**.

For instance below is a first anonymous function that keeps only character variables the dataset.

Ví dụ bên dưới là một hàm ẩn danh đầu tiên mà chỉ giữ các biến ký tự trong bộ số liệu.

Then another function that selects every second observation of our dataset (may be relevant when we have longitudinal data with many records per patient for instance after having ordered by date or visit). In this case, the proper function writing outside dplyr would be function (x) (x%%2 == 0) to apply to the vector containing all row numbers.

Sau đó, một hàm khác chọn mỗi quan sát thứ hai của bộ số liệu (có thể thích hợp khi chúng ta có số liệu dọc với nhiều bản ghi cho một bệnh nhân, ví dụ như ngày đặt lịch khám hoặc lần khám). Trong trường hợp này, việc viết hàm thích hợp bên ngoài gói lệnh dplyr sẽ là function (x) (x%%2 == 0)để áp dụng đối với véc tơ chứa tất cả các số hàng.

A possible base R code for the same task would be:

Một mã trong base R có thể có cùng tác vụ:

***CAUTION:*** Though it is true that using functions can help us with our code, it can nevertheless be time consuming to write some functions or to fix one if it has not been thought thoroughly, written adequately and is returning errors as a result. For this reason it is often recommended to first write the R code, make sure it does what we intend it to do, and then transform it into a function with its three main components as listed above.

***THẬN TRỌNG:*** Mặc dù đúng là việc sử dụng hàm có thể giúp chúng ta viết mã, tuy nhiên, việc viết một số hàm hay sửa hàm có thể tốn thời gian nếu chưa được suy nghĩ thấu đáo, viết đầy đủ và kết quả là báo lỗi. Vì lý do này, đầu tiên chúng ta nên viết mã trong R, đảm bảo nó thực hiện những gì chúng ta dự định và sau đó chuyển nó vào trong một hàm với ba thành phần chính như đã liệt kê bên trên.

## Examples

## Ví dụ

### Return proportion tables for several columns

### Trả về các bảng tỷ lệ cho một số cột

Yes, we already have nice functions in many packages allowing to summarize information in a very easy and nice way. But we will still try to make our own, in our first steps to getting used to writing functions.

Đúng vậy, chúng ta đã có các hàm trong nhiều gói lệnh mà cho phép tóm tắt thông tin một cách dễ dàng và đẹp mắt. Nhưng chúng ta vẫn cố gắng thử tạo ra hàm riêng trong những bước đầu tiên để làm quen với việc viết hàm.

In this example we want to show how writing a simple function would avoid you copy-pasting the same code multiple times.

Trong ví dụ này, chúng tôi muốn chỉ ra cách viết một hàm đơn giản để tránh việc sao chép và dán cùng một mã nhiều lần.

***TIP:*** As shown above, it is very important to comment your functions as you would do for the general programming. Bear in mind that a function’s aim is to make a code ready to read, shorter and more efficient. Then one should be able to understand what the function does just by reading its name and should have more details reading the comments.

***MẸO:*** Như đã trình bày bên trên, điều quan trọng là chú thích cho các hàm như đã làm đối với lập trình chung. Hãy nhớ rằng mục đích của một hàm là làm cho mã sẵn sàng để đọc, ngắn hơn và hiệu quả hơn. Sau đó, người đọc có thể hiểu được hàm này sẽ làm gì chỉ bằng cách đọc tên nó và có thêm thông tin chi tiết khi đọc các chú thích.

A second option is to use this function in another one via a loop to make the process at once:

Tùy chọn thứ hai là sử dụng hàm này trong một hàm khác thông qua vòng lặp để thực hiện quá trình cùng một lúc.

A simpler way could be using the base R “apply” instead of a “for loop” as expressed below:

Một cách đơn giản hơn có thể là sử dụng “apply” trong base R thay vì dùng “for loop” như được trình bày bên dưới.

***TIP:*** R is often defined as a functional programming language and almost anytime you run a line of code you are using some built-in functions. A good habit to be more comfortable with writing functions is to often have an internal look at how the basic functions you are using daily are built. The shortcut to do so is selecting the function name and then clicking onCtrl+F2 or fn+F2 or Cmd+F2 (depending on your computer) .

***MẸO:*** R thường được định nghĩa như một ngôn ngữ lập trình hàm và hầu như bất cứ lúc nào chúng ta thực hiện một dòng mã, chúng ta đang sử dụng một số hàm có sẵn. Một thói quen tốt để cảm thấy thoải mái hơn với việc viết hàm là thường xuyên nhìn bên trong cách các hàm cơ bản được viết mà chúng ta sử dụng hàng ngày. Phím tắt để làm như vậy là chọn tên hàm và sau đó nhấp vào Ctrl+F2, fn+F2 hoặc Cmd+F2 (tùy vào máy tính) .

## Using purrr: writing functions that can be iteratively applied

## Sử dụng gói lệnh purrr: viết các hàm với vòng lặp

### Modify class of multiple columns in a dataset

### Sửa đổi loại biến (class) của nhiều cột trong một bộ số liệu

Let’s say many character variables in the original linelist data need to be changes to “factor” for analysis and plotting purposes. Instead of repeating the step several times, we can just use lapply() to do the transformation of all variables concerned on a single line of code.

***CAUTION:*** lapply() returns a list, thus its use may require an additional modification as a last step.

The same step can be done using map\_if() function from the **purrr** package

Giả sử nhiều biến ký tự trong bộ số liệu gốc linelist cần được thay đổi thành “factor” nhằm mục đích phân tích và vẽ biểu đồ. Thay vì lặp lại bước nhiều lần, chúng ta có thể sử dụng hàm lapply() để chuyển đổi tất cả các biến có liên quan trong một dòng mã.

***THẬN TRỌNG:*** hàm lapply() trả về một danh sách, vì thế, sử dụng danh sách này có thể đòi hỏi một sửa đổi bổ sung như là một bước cuối cùng.

Bước tương tự có thể được thực hiện bằng cách sử dụng hàm map\_if() từ gói lệnh **purrr.**

### Iteratively produce graphs for different levels of a variable

### Tạo biểu đồ với vòng lặp cho các giá trị của biến

We will produce here pie chart to look at the distribution of patient’s outcome in China during the H7N9 outbreak for each province. Instead of repeating the code for each of them, we will just apply a function that we will create.

Ở đây, chúng ta sẽ tạo biểu đồ hình tròn để xem phân bố kết cục của bệnh nhân trong đợt dịch H7N9 cho từng tỉnh tại Trung Quốc. Thay vì lặp lại mã cho từng tỉnh, chúng ta chỉ áp dụng một hàm mà chúng ta sẽ tạo ra.

### Iteratively produce tables for different levels of a variable

### Tạo các bảng với vòng lặp cho các giá trị của biến

Here we will create three indicators to summarize in a table and we would like to produce this table for each of the provinces. Our indicators are the delay between onset and hospitalization, the percentage of recovery and the median age of cases.

Ở đây, chúng ta sẽ tạo ba chỉ số để tóm tắt trong một bảng và chúng ta muốn tạo bảng này theo từng tỉnh. Các chỉ số của chúng ta là khoảng thời gian từ lúc khởi phát bệnh đến lúc nhập viện, tỷ lệ hồi phục (%) và tuổi trung vị của các ca bệnh.

## Tips and best Practices for well functioning functions

## Mẹo và thực hành tốt để các hàm hoạt động tốt

Functional programming is meant to ease code and facilitates its reading. It should produce the contrary. The tips below will help you having a clean code and easy to read code.

Lập trình hàm có nghĩa là để dễ dàng viết và đọc mã. Điều ngược lại có thể xảy ra. Những mẹo bên dưới sẽ giúp chúng ta có một mã gọn gàng và dễ đọc.

### Naming and syntax

### Đặt tên và cú pháp

* Avoid using character that could have been easily already taken by other functions already existing in your environment
* It is recommended for the function name to be short and straightforward to understand for another reader
* It is preferred to use verbs as the function name and nouns for the argument names.
* Tránh sử dụng ký tự có thể đã được sử dụng bởi các hàm khác trong hàm của chúng ta
* Nên đặt tên hàm ngắn gọn và dễ hiểu cho người đọc
* Ưu tiên sử dụng động từ làm tên hàm và danh từ làm tên các đối số.

### Column names and tidy evaluation

### Tên cột và tidy evaluation (lượng giá tính gọn gàng)

If you want to know how to reference *column names* that are provided to your code as arguments, read this [tidyverse programming guidance](https://dplyr.tidyverse.org/articles/programming.html). Among the topics covered are *tidy evaluation* and use of the *embrace* {{ }} “double braces”

For example, here is a quick skeleton template code from page tutorial mentioned just above:

Nếu chúng ta muốn biết cách tham chiếu *tên cột* vào mã của chúng ta như các đối số, hãy đọc [hướng dẫn lập trình của tidyverse](https://dplyr.tidyverse.org/articles/programming.html) này. Trong số các chủ đề, nên đọc *tidy evaluation* và sử dụng *embrace* {{ }} “hai dấu ngoặc móc”

Ví dụ: đây là mã mẫu từ trang hướng dẫn được đề cập bên trên:

### Testing and Error handling

### Kiểm tra và xử lý lỗi

The more complicated a function’s task the higher the possibility of errors. Thus it is sometimes necessary to add some verification within the funtion to help quickly understand where the error is from and find a way t fix it.

* It can be more than recommended to introduce a check on the missingness of one argument using missing(argument). This simple check can return “TRUE” or “FALSE” value.
* Use stop() for more detectable errors.
* As we see when we run most of the built-in functions, there are messages and warnings that can pop-up in certain conditions. We can integrate those in our written functions by using the functions message() and warning().
* We can handle errors also by using safely() which takes one function as an argument and executes it in a safe way. In fact the function will execute without stopping if it encounters an error. safely() returns as output a **list** with two objects which are the results and the error it “skipped”.

We can verify by first running the mean() as function, then run it with safely().

As said previously, well commenting our codes is already a good way for having documentation in our work.

Công việc của hàm càng phức tạp thì khả năng xảy ra lỗi càng cao. Vì thế, đôi khi cần phải thêm một vài kiểm tra bên trong hàm để giúp nhanh chóng biết được lỗi ở đâu và tìm cách sửa nó.

* Rất nên đưa một kiểm tra tính thiếu sót của một đối số bằng cách sử dụng hàm missing(argument). Việc kiểm tra đơn giản này trả về giá trị “ĐÚNG” hoặc “SAI”.
* Sử dụng hàm stop() để dễ phát hiện lỗi hơn.
* Như đã thấy khi chúng ta thực hiện hầu hết các hàm có sẵn, có các thông báo và cảnh báo có thể xuất hiện trong một số điều kiện nhất định. Chúng ta có thể tích hợp chúng trong các hàm đã viết bằng cách sử dụng hàm message() và warning().
* Chúng ta cũng có thể xử lý lỗi bằng cách sử dụng hàm safely() mà lấy một hàm làm đối số và thực hiện nó một cách an toàn. Trên thực tế, hàm sẽ thực hiện mà không dừng lại nếu nó gặp lỗi. Hàm safely() trả về kết quả đầu ra dưới dạng một **danh sách** với hai đối tượng là kết quả và lỗi mà nó đã “bỏ qua”.

Chúng ta có thể kiểm tra bằng cách thực hiện hàm mean() trước, sau đó thực hiện nó với hàm safely().

Như đã nói trước đây, chú thích rõ ràng trong các đoạn mã là một cách tốt để có tư liệu cho các công việc đã thực hiện.

## Resources

## Nguồn

[R for Data Science link](https://r4ds.had.co.nz/functions.html)

[Cheatsheet advance R programming](https://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf)

[Cheatsheet purr Package](https://purrr.tidyverse.org/)

[Video-ACM talk by Hadley Wickham: The joy of functional programming (how does map\_dbl work)](https://youtube.videoken.com/embed/bzUmK0Y07ck)