# Arguments of the function: one per line

Example:

**fun\_info <- function(**

**data,**

**n = NULL,**

**warn.print = TRUE**

**){**

And not

**fun\_info <- function(data, n = NULL, warn.print = TRUE){**

# In the description section (large part with lines starting by #)

Check that all these categ are here

*#' @title*

*#' @description*

*#' @param*

*#' @returns*

*#' @details*

*#' @examples*

*###' @importFrom # si ::*

*###' @seealso # page html, à regarder*

*#' @export*

Vérifier que c’est bien tout comme la fonction fun\_check

In the argument section, update all argument description, saying, character vector, etc, as in fun\_check ou fun\_gg\_boxplot. Voir la partie avec fun\_check length = 1 ou pas

dans chaque fonction vérifier si single ou pas

In the return section, update all as in fun\_gg\_boxplot

In the example section, 1) copy-paste all the examples in a new file, named *examples\_info.R* for *info.R*, 2) keep a single example in the example section.

In the debbuging section, keep only one line

# à réflichir pour les exemples

# Check that all these comment categs are in the code of the function

    # function name

    # end function name

    # package checking

    # end package checking

    # required function checking

    # end required function checking

    # argument primary checking

    # arg with no default values

    # end arg with no default values

    # argument checking with fun\_check()

    # end argument checking with fun\_check()

    # check with r\_debugging\_tools

    # source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.7/r\_debugging\_tools-v1.7.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()

    # end check with r\_debugging\_tools

    # end argument primary checking

    # second round of checking and data preparation

    # management of NA arguments

    # end management of NA arguments

    # management of NULL arguments

    # end management of NULL arguments

    # code that protects set.seed() in the global environment

    # end code that protects set.seed() in the global environment

    # warning initiation

    # end warning initiation

    # other checkings

    # end other checkings

    # reserved word checking to avoid bugs (e.g., names of dataframe columns used in this function)

    # end reserved word checking to avoid bugs (e.g., names of dataframe columns used in this function)

    # end second round of checking and data preparation

    # main code

    # end main code

    # output

    # warning output

    # end warning output

    # end output

# check also the "must" written in red below

*# function name*

**#** *must have* **arg.user.setting** *inside*

*# end function name*

*# management of NA arguments*

**#** *must have* **arg.user.setting** *inside*

*# end management of NA arguments*

# check avec debugging\_tools\_for\_r\_dev-v1.7

2) Cette partie, tu sais, c'est celle où il faut enlever le # au début. Sauver le fichier, sourcer le fichier, et éxecuter la fonction. Ca va alors vérifier quelques trucs.

    # check with r\_debugging\_tools

    # source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.7/r\_debugging\_tools-v1.7.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()

    # end check with r\_debugging\_tools

Il faut ensuite remettre le # et sauver le fichier.

Tu peux forker :

<https://gitlab.pasteur.fr/gmillot/debugging_tools_for_r_dev>

Ensuite le cloner sur to Git\_project local

et remplacer :

    # source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.7/r\_debugging\_tools-v1.7.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()

Par le bon path dans chaque fonction. PAr exemple:

    # source("C:/Users/yhan/Documents/Git\_project/debugging\_tools\_for\_r\_dev-v1.7/r\_debugging\_tools-v1.7.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()

# Check des warnings

Regarder si dans la fonction, il y a des warnings (faire une recherche "warn")

Si non, écrire :

# warning initiation

# end warning initiation

Tous les messages d'erreurs sont:

stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), *call.* = FALSE)

La fin est (section vide):

      # warning output

      # end warning output

A ajouter dans la section

    # output

    # end output

Si oui :

# warning initiation

ini.warning.length <- options()$warning.length

options(*warning.length* = 8170)

warn <- NULL

warn.count <- 0

# end warning initiation

Tous les messages d'erreurs avant # warning initiation sont :

stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), *call.* = FALSE)

Tous les messages d'erreurs après # end warning initiation sont :

stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), *call.* = FALSE)

La fin est:

      # warning output

      if(warn.print == TRUE & ! is.null(warn)){

        on.exit(warning(paste0("FROM ", function.name, ":\n\n", warn), *call.* = FALSE))

      }

      on.exit(*exp* = options(*warning.length* = ini.warning.length), *add* = TRUE)

      # end warning output

A ajouter au début de la section

    # output

    # end output

Attention à cette section de code que j'avais écrite hier (remise dessous):

      # warning output

      if(warn.print == TRUE & ! is.null(warn)){

        on.exit(warning(paste0("FROM ", function.name, ":\n\n", warn), *call.* = FALSE))

      }

      on.exit(*exp* = options(*warning.length* = ini.warning.length), *add* = TRUE)

      # end warning output

Ce code est valable s'il existe un argument warn.print dans la fonction (fun\_segmentation l'a par exemple).

Mais si la fonction n'a pas l'argument warn.print, il faut écrire :

      # warning output

      if( ! is.null(warn)){

        on.exit(warning(paste0("FROM ", function.name, ":\n\n", warn), *call.* = FALSE))

      }

      on.exit(*exp* = options(*warning.length* = ini.warning.length), *add* = TRUE)

      # end warning output

# Checks avec fun\_check:

Il faudrait vérifier que dans

# argument checking with fun\_check()

il n'y a que les tests des arguments avec fun\_ckeck.

Si non, me le dire pour déplacer les autres checks.

# use fun\_test() for each argument of the function

Il faut d'abord qu'on créer un bon object de test, mieux que t26.

Ensuite faire un test pour chaque argument séparément