Logiciel R et programmation

Fonctions

Ewen Gallic
Université de Rennes 1, 2014 - 2015

Source : http://dilbert.com/strips/comic/2007-11-26/

Définition d'une fonction

La syntaxe est la suivante :

```
name <- function(arguments) expression
```

`name` : nom donné à la fonction (attention à respecter les contraintes de nommage) ;

`arguments` : les paramètres de la fonction ;

`expression` : le corps de la fonction.

Appel d'une fonction

Appel par son nom;

Paramètres passés entre parenthèses, juste derrière le nom de la fonction.

```
name(argument_1, argument_2)
```

Exemple

Créons la fonction `carre()`, qui retourne le carré d'un nombre.

```
carre <- function(x) x^2
carre(2)
```

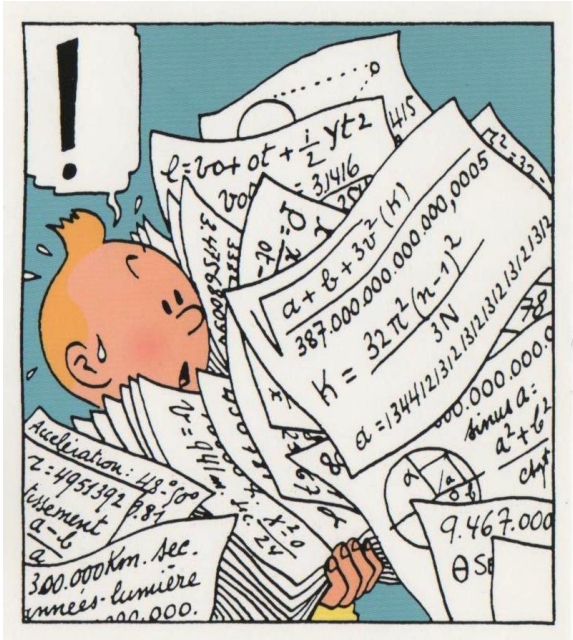
```
## [1] 4
```

```
carre(-3)
```

```
## [1] 9
```

Créer une fonction qui, pour deux paramètres x et y retourne le résultat suivant :

$$\frac{xy}{x^2 + y^2}.$$



Source : Tintin - L'étoile mystérieuse

Structure d'une fonction

Excepté pour les fonctions primitives du package `base`, les fonctions sont composées de trois parties :

- une liste de paramètres ;
- un corps, contenant du code exécuté lors de l'appel à la fonction ;
- un environnement, qui définit l'endroit où sont stockées les variables.

L'accès (et la modification) peut se faire avec les fonctions :

- `formals()` pour les paramètres ;
- `body()` pour le corps ;
- `environment()` pour l'environnement.

Le corps d'une fonction

Une seule instruction dans le cas le plus simple ;

Entourer d'accolades pour mettre plusieurs instructions ;

Le résultat est la valeur de la dernière commande contenue dans le corps de la fonction.

```
f <- function(x) {
  x^2
  y <- x
  y
}
f(2)
```

```
## [1] 2
```

Le corps d'une fonction

La fonction `return()` permet de retourner un résultat autre part qu'à la dernière ligne ;

Cela peut être utile dans le cas de conditions à l'intérieur du corps ;

Mais il est déconseillé d'utiliser `return()` en dernière ligne (inutile, et perturbant).

```
f <- function(x) {
  return(x^2)
  # Un commentaire de dernière ligne
}
f(2)
```

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

Le corps d'une fonction

On peut retourner une liste, ce qui est extrêmement pratique (pas de conversion automatique !)

```
stat_des <- function(x) {
  list(moyenne = mean(x), ecart_type = sd(x))
}
x <- runif(10)
stat_des(x)
```

```
## $moyenne
## [1] 0.5566102
##
## $ecart_type
## [1] 0.2913794
```

Le corps d'une fonction

On peut éviter l'affichage du résultat dans la console en utilisant la fonction `invisible()` ;

```
stat_des_2 <- function(x) {
invisible(list(moyenne = mean(x), ecart_type = sd(x)))
}
x <- runif(10)
stat_des_2(x)
str(stat_des_2(x))
```

```
## List of 2
##  $ moyenne   : num 0.537
##  $ ecart_type: num 0.307
```

```
stat_des_2(x)$moyenne
```

```
## [1] 0.5370646
```

Le corps d'une fonction

L'affichage d'un résultat caché peut être forcé, en utilisant les parenthèses :

```
(stat_des_2(x))
```

```
## $moyenne
## [1] 0.5370646
##
## $ecart_type
## [1] 0.3065536
```

Le corps d'une fonction : remarque

Si la dernière instruction est une assignation, le résultat est invisible ;

Mais la valeur d'assignation est bien retournée.

```
f <- function(x){
  res <- x^2
}
f(2)
(f(2))
```

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

```
x <- f(2)
x
```

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

Les paramètres d'une fonction

Jusqu'ici, un seul paramètre dans les fonctions ;

Si plusieurs : les séparer par une virgule.

Exemple : soit la fonction de production $Y(L, K, M)$, qui dépend du nombre de travailleurs L et de la quantité de capital K, et du matériel M, telle que $Y(L, K, M) = L^{0.3} K^{0.5} M^{2}$. Cette fonction pourra s'écrire, en R de la manièere suivante :

```
production <- function(l, k, m) l^(0.3) * k^(0.5) * m^(0.2)
```

Appel sans nom

Si on nous donne les valeurs $L = 60$ et $K = 42$ et $M = 40$, on peut en déeduire la production :

```
production(60, 42, 40)
```

```
## [1] 46.28945
```

Les paramètres n'ont pas été nommés, `R` fait donc en fonction de l'ordre donné.

Appel sans nom

R cherche d'abord les paramètres nommés, puis complète en fonction de la position :

```
production(k = 42, m = 40, l = 60)
```

```
## [1] 46.28945
```

```
production(k = 42, 60, 40)
```

```
## [1] 46.28945
```

Paramètres effectifs

Paramètre : paramètre de la fonction, utilisé dans le corps ;

Paramètre : valeur que l'on donne au paramètre formel ;

On peut donner une valeur par défaut au paramètre formel à l'aide du signe **=** ;

Si l'utilisateur omet ce paramètre lors de l'appel, il prendra la valeur automatique ;

Sinon, la valeur attribuée lors de l'appel prévaudra.

Paramètres effectifs

```
# On propose de définir la valeur du capital à 42 par défaut
production_2 <- function(l, m, k = 42) l^(0.3) * k^(0.5) * m^(0.2)
production_2(l = 42, m = 40)
```

```
## [1] 41.59216
```

```
production_2(l = 42, m = 40, k = 2)
```

```
## [1] 9.076152
```

1. Créer une fonction, qui étant donné un quantile x, une espérance μ et un écart-type σ, retourne la densité de la loi normale. Comparer avec la fonction `dnorm()`.
2. Reprendre la fonction de la question 1, et donner les valeurs par défaut 1 et 0 pour l'espérance et l'écart-type respectivement.

Appel avec des noms partiels

R permet de ne pas saisir le nom complet des paramètres ;

Cependant, il ne faut pas qu'il y ait d'ambiguïté !

```
f <- function(premier, second, troisieme) premier + second + troisieme
f(p = 1, s = 2, t = 3)
```

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

```
# Problème d'ambiguïté
f <- function(texte, nombre, nom) print(nom)
f("hello", 2, no = 3)
```

```
## Error: l'argument 3 correspond à plusieurs arguments formels
```

Appel avec des noms partiels

Attention, si le paramètre `...` figure dans la liste des paramètres de la fonction que l'on utilise, le nommage partiel ne fonctionne pas.

Fonctions sans paramètres

Il est possible de créer une fonction sans paramètre.

```
f <- function() sample(letters, size = 10, replace = TRUE)
```

Le paramètre spécial `...`

Le paramètre `...` sert à indiquer que la fonction peut admettre d'autres paramètres que ceux qui sont définis ;

Utile pour passer un paramètre à une fonction à l'intérieure d'une autre.

```
f <- function(...) names(list(...))
f(premier = 1, second = 2)
```

```
## [1] "premier" "second"
```

Voir la fonction `sum()` par exemple.

Le paramètre spécial `...`

Attention, une mauvaise orthographe est fatale quand le paramètre `...` est présent !

Le paramètre mal écrit sera passé à `...` et aucune erreur ne sera retournée ;

On comprend mieux pourquoi l'abréviation n'est pas possible quand `...` est présent.

```
sum(3, NA, 4, na.rm = TRUE)
```

```
## [1] 7
```

```
sum(3, NA, 4, an.rm = TRUE) # Mauvaise écriture
```

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

```
sum(3, NA, 4, na = TRUE)  # Abréviation
```

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

La portée d'une fonction

Lors de l'appel d'une fonction, le corps est interprété ;

Les variables créées, modifiées dans le corps ne vient qu'à l'intérieur de la fonction ;

On parle de portée des variables ;

Une variable locale (qui vit dans la fonction) peut avoir le même nom qu'une variable globale (qui vit dans l'espace de travail) ;

La portée d'une fonction

```
# Définition d'une variable globale
valeur <- 1

# Définition d'une variable locale à la fonction f
f <- function(x){
  valeur <- 2
  nouvelle_valeur <- 3
  print(paste0("valeur vaut : ",valeur))
  print(paste0("nouvelle_valeur vaut : ",valeur))
  x + valeur
}
```

La portée d'une fonction

```
f(3)
```

```
## [1] "valeur vaut : 2"
## [1] "nouvelle_valeur vaut : 2"
```

```
## [1] 5
```

```
# valeur n'a pas été modifiée
valeur
```

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

```
# nouvelle_valeur n'existe pas en dehors de f()
nouvelle_valeur
```

```
## Error: objet 'nouvelle_valeur' introuvable
```

La portée d'une fonction

Les variables sont définies dans des environnements ;

Ces environnements sont imbriqués les uns dans les autres ;

Si `R` ne trouve pas une variable dans l'environnement courant, il va chercher dans le (ou les) niveau(x) supérieur(s) ;

Ainsi, si une variable n'est pas définie dans le corps d'une fonction, `R` essaie de la trouver dans un environnement parent.

```
valeur <- 1
f <- function(x){
  x + valeur
}
f(2)
```

```
## [1] 3
```

La portée d'une fonction

Pour rendre accessible dans l'environnement global une variable créée dans le corps d'une fonction, on peut utiliser le symbole `<<-` ;

La fonction `assign()` est plus complète, puisqu'elle permet de choisir l'environnement d'affectation.

La portée d'une fonction

```
rm(x)
f <- function(x){
  x <<- x + 1
}
f(1)
x
```

```
## [1] 2
```

La portée d'une fonction

```
rm(x)
f <- function(x){
  # envir = .GlobalEnv signifique que l'on veut définir dans l'environnement global
  assign(x = "x", value = x + 1, envir = .GlobalEnv)
}
f(4)
x
```

```
## [1] 5
```