Les classes R6 sous R

Programmation orientée objet sous R

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

R est un « vrai » langage de programmation. Il possède plusieurs mécanismes de classes (S3, S4, RC). Mais leurs spécificités peuvent dérouter les férus de programmation orientée objet (POO).

Le package R6 (Winston Chang) introduit un modèle plus conforme à la pratique de la POO des autres langages, avec des notions familières aux programmeurs (portée des membres, constructeur/destructeur…). Il est possible de produire du code plus « élégant », mieux structuré.

Objectifs

- Meilleure organisation du programme, plus de lisibilité, plus de réutilisabilité,… la maintenance est facilitée.
- Meilleure cohérence des packages : objet(s) + fonctions manipulant ce(s) type(s) d'objet(s).

Problème à traiter : corrélation entre 2 vecteurs

Traiter un problème de corrélation entre 2 vecteurs, on doit stocker plusieurs informations.

```
#vecteur x
x <- c(18,19,20,21,22,23,24,25,26,27,28,29)

#vecteur y
y <- c(76.1,77,78.1,78.2,78.8,79.7,79.9,81.1,81.2,81.8,82.8,83.5)

#effectif
n <- length(x)

#correlation
cor.p <- cor(x,y)
print(cor.p)

#t pour test de significativité
cor.t <- cor.p/sqrt((1-cor.p^2)/(n-2))
print(cor.t)

#p-value test de significativité
cor.pvalue <- 2.0*pt(abs(cor.t),n-2,lower.tail=F)
print(cor.pvalue)
```

Objectif : rassembler ces informations dans un tout cohérent.

LE MODÈLE DE CLASSE R6

Définition d'une classe R6 - Instanciation

Package : Il faut installer (une fois) et charger (à chaque fois) le package R6

Déclaration: Une classe R6 correspond à une structure spécifique

```
#charger la librairie
library(R6)

#version du package > 1.0.1 -- classes portables
packageVersion("R6") #2.2.2 pour ce support

#classe très simple avec 3 champs publics
Correlation <- R6Class("Correlation",
  #membres publics
  public = list(
  #champs
  r = NA,
  t = NA,
  n = NA
  )
)

#instanciation
cr <- Correlation$new()

#affectation aux champs
cr$r <- cor.p
cr$t <- cor.t
cr$n <- n

#affichage
print(cr)
```

La version du package est importante. A partir de 1.0.1, les classes sont par défaut « portables ».

La classe est déclarée avec le mot-clé R6Class. On doit définir une portée des membres de la classe. Les champs sont énumérés et initialisés.

La méthode new() permet d'instancier l'objet.

Les champs publics sont par défaut en lecture et écriture. On utilise $ pour accéder aux membres de l'objet.

```
> print(cr)
<Correlation>
  Public:
    clone: function (deep = FALSE)
    n: 12
    r: 0.994366098146598
    t: 29.6646500266205
```

Implémentation d'un "constructeur"

Il est possible (mais pas obligatoire) de définir un constructeur. Son nom est forcément initialize(). Il peut prendre différents paramètres. Il est automatiquement appelé par la méthode new() à l'instanciation.

```
#classe avec constructeur
Correlation <- R6Class("Correlation",
  #membres publics
  public = list(
  #champs
  r = NA,
  t = NA,
  n = NA,
  #constructeur
  initialize = function(prmX,prmY){
    if (length(prmX) != length(prmY) || (length(prmX) < 2)){
      stop("Longueurs des vecteurs incohérentes")
    } else
    {
      #accès aux champs dans les méthodes de la classe
      self$n <- length(prmX)
      self$r <- cor(prmX,prmY)
      self$t <- self$r/sqrt((1-self$r^2)/(self$n-2))
    }
  }
  )
)

#instanciation avec new()
#qui appelle automatiquement le constructeur
#avec 2 paramètres : vecteurs x et y
cr <- Correlation$new(x,y)
print(cr)
```

Pour accéder aux champs publics au sein des méthodes, nous utilisons self$...

```
<Correlation>
  Public:
    clone: function (deep = FALSE)
    initialize: function (prmX, prmY)
    n: 12
    r: 0.994366098146598
    t: 29.6646500266205
```

Implémentation d'un destructeur

Il est possible (mais pas obligatoire) de définir un destructeur. Son nom est forcément finalize(). Il ne prend pas de paramètres. Il est appelé à l'initiative du *garbage collector*, et non pas à la suppression de la référence.

```
#classe avec destructeur
Correlation <- R6Class("Correlation",
   #membres publics
   public = list(
     #champs
     r = NA,
     t = NA,
     n = NA,
     #constructeur
     initialize = function(prmX,prmY){
      ... voir le code de la page précédente ...
     },
     #destructeur
     finalize = function(){
       cat("<<< destructeur a été appelé ! >>>\n")
     }
   )
)

#instanciation
cr <- Correlation$new(x,y)
print(cr)

#suppression de la référence
rm(cr)
print("objet removed par rm")

#appel explicite du garbage collector
gc()
```

Plus que la suppression d'objets, il peut être utile pour la fermeture d'une connexion, ou encore de fichiers temporaires, etc.

L'appel du destructeur intervient à l'initiative du ramasse-miettes et non lors de l'appel de rm()

```
> print(cr)
<Correlation>
  Public:
    clone: function (deep = FALSE)
    finalize: function ()
    initialize: function (prmX, prmY)
    n: 12
    r: 0.994366098146598
    t: 29.6646500266205
> #destruction
> rm(cr)
> print("objet removed par rm")
[1] "objet removed par rm"
> #appel du garbage collector
> gc()
<<< destructeur a été appelé ! >>>
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 481371 25.8     940480 50.3   750400 40.1
Vcells 873613  6.7    1650153 12.6  1213204  9.3
```

Implémentation d'une méthode

La déclaration et l'implémentation des méthodes supplémentaires est simple. La gestion des paramètres est conforme aux standards de R.

L'appel de la méthode obéit au schéma **objet**$méthode()

```
#classe avec méthode publique "tester"
Correlation <- R6Class("Correlation",
  #membres publics
  public = list(
  #champs
  r = NA,
  t = NA,
  n = NA,
  #constructeur
  initialize = function(prmX, prmY) {
   ... voir constructeur des pages précédentes ...
  },
  #test H0 : r = 0 - implémentation
  tester = function(sens){
    #unilateral à gauche
    if (sens == "gauche"){
      return(pt(self$t,self$n-2,lower.tail=TRUE))
    }
    #unilateral à droite
    if (sens == "droite"){
      return(pt(self$t,self$n-2,lower.tail=FALSE))
    }
    #bilatéral
    if (sens == "bilateral"){
      return(2*pt(abs(self$t),self$n-2,lower.tail=FALSE))
    }
    #autrement...
    return(NA)
  }
 )
)

#instanciation
cr <- Correlation$new(x,y)

#appel de la méthode
cr$tester("bilateral")
```

```
> #instanciation
> cr <- Correlation$new(x,y)
> cr$tester("bilateral")
[1] 4.428071e-11
```

Accesseurs, mutateurs, propriétés

PORTÉE DES MEMBRES

Portée – Membres publics et privés

```
Correlation <- R6Class("Correlation",
   #membres publics
   public = list(
     #constructeur
     initialize = function(prmX,prmY){
       if (length(prmX) != length(prmY) || (length(prmX) < 2)){
         stop("Longueurs des vecteurs incohérentes")
       } else
       {
         #accès aux champs privés de la classe
         private$n <- length(prmX)
         private$r <- cor(prmX,prmY)
         private$t <- self$r/sqrt((1-self$r^2)/(self$n-2))
       }
     },
     #accesseurs publics
     correlation = function(){
       return(private$r)
     },
     tstudent = function(){
       return(private$t)
     },
     effectif = function(){
       return(private$n)
     }
   ),
  #membres privés
  private = list(
    #champs
    r = NA,
    t = NA,
    n = NA
  )
)

#instanciation
cr <- Correlation$new(x,y)

#affichage de la correlation
cr$correlation()

#tentative écriture champ privé
cr$r <- 10
```

On peut définir la portée des membres de la classe : privé ou public.

L'accès aux membres privés au sein des méthode se fait avec le mot-clé private (c.-à-d. private$membre)

La tentative d'accès à un membre privé sur l'instance provoque une erreur bien évidemment (en lecture et en écriture)

```
> cr$r <- 10
Error in cr$r <- 10 : cannot add bindings to a locked environment
```

Mécanisme des propriétés en lecture et écriture (1/2)

```
#utilisation des propriétés - les "active bindings"
Correlation <- R6Class("Correlation",
   #membres publics
   public = list(
     #constructeur avec 4 paramètres maintenant
     initialize = function(prmX,prmY,nomX="",nomY=""){
       if (length(prmX) != length(prmY) || (length(prmX) < 2)){
         stop("Longueurs des vecteurs incohérentes")
       } else
       {
         #accès aux champs dans les méthodes de la classe
         private$n <- length(prmX)
         private$r <- cor(prmX,prmY)
         private$t <- self$r/sqrt((1-self$r^2)/(self$n-2))
         private$FNameX <- nomX
         private$FNameY <- nomY
       }
     }
   ),
   #propriétés
   active = list(
     #nom de variable X (lecture/écriture)
     NameX = function(value){
       #si pas de value => lecture
       if (missing(value) == TRUE){
         return(private$FNameX)
       } else
       {
         #si value existe ==>
         #écriture de la propriété
         private$FNameX <- value
       }
     }
   ),
   #membres privés
   private = list(
     #champs
     n = NA,
     r = NA,
     t = NA,
     FNameX = "",
     FNameY = ""
   )
)
```

Les propriétés (*active bindings*) sont membres qui se comportent comme des champs publics mais dont l'accès en lecture et écriture déclenche l'exécution d'une fonction qui joue le rôle d'accesseur et mutateur. **value** est une variable qui transmet la valeur transmise en affectation (comme en C#).

Dans cet exemple, **NameX** est une propriété qui permet d'accéder en lecture et écriture au champ privé **FNameX**.

Mécanisme des propriétés en lecture et écriture (2/2)

Instanciation et accès aux champs privés via les propriétés.

```
#instanciation
cr <- Correlation$new(x,y,"V1","V2")

#lecture
cr$NameX

#affectation
cr$NameX <- "toto"

#lecture de nouveau
cr$NameX

#print
cr
```

L'accès en lecture de NameX déclenche l'exécution de la fonction associée. Le paramètre value est vide. La valeur de FNameX est donc simplement retournée.

L'affectation de la valeur « toto » à NameX déclenche l'exécution de la fonction associée. Le paramètre value contient « toto » qui sera affecté à FNameX.

```
> #instanciation
> cr <- Correlation$new(x,y,"V1","V2")
>
> #lecture
> cr$NameX
[1] "V1"
>
> #affectation
> cr$NameX <- "toto"
>
> #lecture de nouveau
> cr$NameX
[1] "toto"
>
> #print
> cr
<Correlation>
  Public:
    clone: function (deep = FALSE)
    initialize: function (prmX, prmY, nomX = "", nomY = "")
    NameX: active binding
  Private:
    FNameX: toto
    FNameY: V2
    n: 12
    r: 0.994366098146598
    t:
```

Propriétés en lecture seule

```
Correlation <- R6Class("Correlation",
   #membres publics
   public = list(
     #constructeur
     initialize = function(prmX,prmY,nomX="",nomY=""){
      ... voir page précédente ...
     }
   ),
   #propriétés
   active = list(
     #nom de variable X (lecture/écriture)
     NameX = function(value){
       #si pas de value => lecture
       if (missing(value) == TRUE){
         return(private$FNameX)
       } else
       {
         #écriture de la propriété
         private$FNameX <- value
       }
     },
     #propriété corrélation en lecture seule
     #pas de paramètre value
     correlation = function(){
       return(private$r)
     }
   ),
   #membres privés
   private = list(
    ... voir page précédente ...
   )
)

#instanciation
cr <- Correlation$new(x,y,"V1","V2")

#affichage de la propriété, pas de () dans l'appel
#appel de l'accesseur en sous-main en réalité
cr$correlation

#propriété en lecture seule ==> erreur
cr$correlation <- 0.0
```

A une propriété en lecture seule est associé un accesseur sans le paramètre value.

```
> #propriété en lecture seule
> cr$correlation <- 0.0
Error in (function ()  : unused argument (quote(0))
```

FONCTION GÉNÉRIQUE PRINT

Programmer la fonction print()

print() est une méthode de classe qui n'est pas comme les autres. Elle peut se comporter comme une fonction générique.

```
Correlation <- R6Class("Correlation",
   #membres publics
   public = list(
     #constructeur
     initialize = function(prmX,prmY,nomX="",nomY=""){
      ... voir les pages précédentes ...
     },
     #méthode print
     print = function(){
       cat("X :", private$FNameX,"\n")
       cat("Y :", private$FNameY,"\n")
       cat("Correlation :", private$r,"\n")
     }
   ),
   #membres privés
   private = list(
     #champs
     n = NA,
     r = NA,
     t = NA,
     FNameX = "",
     FNameY = ""
   )
)

#instanciation
cr <- Correlation$new(x,y,"V1","V2")

#appel usuel comme méthode
cr$print()

#ou appel comme fonction générique !!!
print(cr)
```

```
> #ou appel comme fonction générique
> print(cr)
X : V1
Y : V2
Correlation : 0.9943661
```

HÉRITAGE

Héritage de classes

Organiser les classes en structures hiérarchiques permet de maximiser la réutilisabilité du code. Mais cela implique une réflexion (modélisation) en amont.

Regression est un héritier de Correlation.

super$... permet d'accéder aux membres de la classe ancêtre.

```
Regression <- R6Class("Regression",
  inherit = Correlation, #indication de la classe ancêtre
  public = list(
    #constructeur
    initialize = function(prmX,prmY,nomX="",nomY=""){
      #appel du constructeur de l'ancêtre
      super$initialize(prmX,prmY,nomX,nomY)
      #et régression
      private$FReg = lm(prmY ~ prmX)
    },
    #surcharge de print
    print = function(){
      super$print()
      cat("Reg. : Pente = ",private$FReg$coefficients[2],"\n")
      cat("Reg. : Constante = ",private$FReg$coefficients[1],"\n")
    }
  ),
  #champs privés supplémentaires
  private = list(
    #champ régression
    FReg = NULL
  ),
  #propriétés supplémentaires
  active = list(
    #accesseur pour le champ supp.
    regression = function(){
      return(private$FReg)
    }
  )
)

#instanciation
reg <- Regression$new(x,y,"V1","V2")

#affichage avec print surchargé
print(reg)

#accès à la propriété
print(reg$regression)
```

```
> print(reg)
X : V1
Y : V2
Correlation : 0.9943661
Reg. : Pente =  0.634965
Reg. : Constante =  64.92832
> print(reg$regression)

Call:
lm(formula = prmY ~ prmX)

Coefficients:
(Intercept)         prmX
     64.928        0.635
```

CONCLUSION

Le mécanisme des classes nous permet de programmer plus proprement, et par conséquent plus efficacement, sous R.

Les modèles de classes « standards » sous R (S_3, S_4, RC) ne sont pas évidents à manipuler, en grande partie parce qu'ils dérogent aux règles usuelles de la POO (programmation orientée objet).

Le modèle de classes R6 permet de pallier ces insuffisances en proposant un cadre très proche de ce que l'on peut trouver dans les autres langages de programmation.

Après, l'enjeu est de ne pas dérouter les aficionados de R….

W. Chang, “R6: Classes with Reference Semantics”, https://cran.r-project.org/web/packages/R6/ et surtout https://cran.r-project.org/web/packages/R6/vignettes/Introduction.html

W. Chang, “The R6 Class System”, in UseR!, Brussels 2017, https://www.youtube.com/watch?v=3GEFd8rZQgY

D. Smith, “The R6 Class System”, in R-bloggers, July 2017, https://www.r-bloggers.com/the-r6-class-system/

Conclusion - R est magique

De la documentation à profusion (**n'achetez jamais des livres sur R**)

Site du cours
http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Programmation R
http://www.duclert.org/

Quick-R
http://www.statmethods.net/

POLLS (Kdnuggets)
Data Mining / Analytics Tools Used
(R, 2nd ou 1er depuis 2010)
What languages you used for data mining / data analysis?
http://www.kdnuggets.com/polls/2013/languages-analytics-data-mining-data-science.html
(Août 2013, langage R en 1ère position)

Article New York Times (Janvier 2009)
"**Data Analysts Captivated by R's Power**" -
http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=1