R: Langage

Eric Marcon

02 mai 2018

Architecture

R-core

Fonctions primitives et structures de données de base.

Exemples : fonction `sumc` et données de `typematrix`‘:

```
pryr::otype(sum)
```

```
## [1] "base"
```

```
pryr::otype(matrix(1))
```

```
## [1] "base"
```

S3

Langage orienté objet.

Classes déclaratives.

```
MonPrenom <- "Eric"
class(MonPrenom) <- "Prenom"
```

S3 - Génériques

Les méthodes sont liées aux fonctions, pas aux objets.

```
# Affichage par défaut
MonPrenom
```

```
## [1] "Eric"
## attr(,"class")
## [1] "Prenom"
```

```
print.Prenom <- function(x) cat("Le prénom est",
    x)
# Affichage modifié
MonPrenom
```

```
## Le prénom est Eric
```

S3 - print

`print` est une méthode générique ("un générique") déclaré dans base.

```
`?`(print)
```

Son code se résume à une déclaration `UseMethod("print")`:

```
print
```

```
## function (x, ...)
## UseMethod("print")
## <bytecode: 0x000000001bd964f0>
## <environment: namespace:base>
```

S3 - print

Il existe beaucoup de méthodes pour `print`:

```
head(methods("print"))
```

```
## [1] "print.acf"     "print.AES"
## [3] "print.anova"   "print.aov"
## [5] "print.aovlist" "print.ar"
```

Chacune s'applique à une classe. `print.default` est utilisée en dernier ressort et s'appuie sur le type (R de base), pas la classe (S3).

```
typeof(MonPrenom)
```

```
## [1] "character"
```

S3 - Héritage

Un objet peut appartenir à plusieurs classes.

```
class(MonPrenom) <- c("PrenomFrancais",
    "Prenom")
inherits(MonPrenom, what = "PrenomFrancais")
```

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

```
inherits(MonPrenom, what = "Prenom")
```

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

S3 - Héritage

Le générique cherche une méthode pour chaque classe, dans l'ordre.

```
print.PrenomFrancais <- function(x) cat("Prénom franç
    x)
MonPrenom
```

```
## Prénom français: Eric
```

S3 - Résumé

S3 est le langage courant de R.

Presque tous les packages sont écrits en S3.

Les génériques sont partout mais passent inaperçu:

```
library("entropart")
```

```
## Loading required package: ggplot2
```

```
.S3methods(class = "SpeciesDistribution")
```

```
## [1] autoplot plot
## see '?methods' for accessing help and source code
```

S4

S4 structure les classes.

```
setClass("Personne", slots = list(Nom = "character",
    Prenom = "character"))
Moi <- new("Personne", Nom = "Marcon",
    Prenom = "Eric")
```

S4 - Méthodes

Les méthodes appartiennent toujours aux fonctions:

```
setMethod("print", signature = "Personne",
    function(x, ...) {
        cat("La personne est:", x@Prenom,
            x@Nom)
    })
```

```
## [1] "print"
```

```
print(Moi)
```

```
## La personne est: Eric Marcon
```

S4 - Résumé

S4 est plus rigoureux que S3.

Quelques packages sur CRAN : Matrix, stats4,... et beaucoup sur Bioconductor.

RC

RC a été introduit dans R 2.12 (2010) dans le package *methods*.

Les méthodes appartiennent aux classes, comme en C++.

```
PersonneRC <- setRefClass("PersonneRC",
    fields = list(Nom = "character",
        Prenom = "character"), methods = list(print =
        Nom)))
MoiRC <- new("PersonneRC", Nom = "Marcon",
    Prenom = "Eric")
MoiRC$print()
```

```
## Eric Marcon
```

S6

S6 perfectionne RC mais n'est pas inclus dans R.

Les attributs et les méthodes peuvent être publics ou privés.

Une méthode `initialize()` est utilisée comme constructeur.

```
library(R6)
PersonneR6 <- R6Class("PersonneR6",
    public = list(Nom = "character",
        Prenom = "character", initialize = function(N
            Prenom = NA) {
            self$Nom <- Nom
            self$Prenom <- Prenom
        }, print = function() cat(Prenom,
            Nom)))
MoiR6 <- PersonneR6$new(Nom = "Marcon",
    Prenom = "Eric")
MoiRC$print()
```

RC et S6 - Résumé

Très peu utilisés, plutôt considérés comme un exercice de style.

S6 permet de programmer rigoureusement en objet.

Les performances sont inférieures à celles de S3.