Ce que j'adore dans R : 5 fonctionnalités qui séduisent un développeur Python

Joseph Barbier[1]

Résumé

R possède de nombreuses particularités qui peuvent sembler étranges pour un développeur Python, mais qui rendent le langage particulièrement puissant et expressif. Dans cette présentation, je montrerai 5 fonctionnalités que j'apprécie particulièrement et qui peuvent transformer votre manière de coder en R.

Vous découvrirez comment créer des opérateurs personnalisés, surcharger des fonctions, utiliser la métaprogrammation, tirer parti du système S3 et profiter de la lazy evaluation pour des arguments par défaut intelligents.

Mots-clés (3 à 5) : R - Python - S3 - Métaprogrammation - Lazy Evaluation

Développement

1. Créer ses propres opérateurs

En R, toute fonction entourée de backticks et de `%` devient un opérateur infixe. Cela permet de créer des DSL clairs et intuitifs.

```
`%within%` <- function(x, range) x >= range[1] & x <= range[2]

5 %within% c(1, 10)
```

```
[1] TRUE
```

```
12 %within% c(1, 10)
```

```
[1] FALSE
```

[1]Yellow Sunflower, joseph@ysunflower.com

2. Surcharge d'opérateurs

Grâce aux méthodes S3, vous pouvez redéfinir le comportement des opérateurs pour vos objets personnalisés.

```
weight <- function(x, unit) structure(x, unit = unit, class = "unit_val")

`+.unit_val` <- function(a, b) {
    if (inherits(b, "unit_val")) {
        b <- weight(unclass(b), attr(a, "unit"))
    }
    weight(unclass(a) + unclass(b), attr(a, "unit"))
}

x <- weight(5, "kg")
y <- weight(300, "g")
z <- weight(2, "kg")
x + y
```

```
[1] 305
attr(,"unit")
[1] "kg"
attr(,"class")
[1] "unit_val"
```

```
x + z
```

```
[1] 7
attr(,"unit")
[1] "kg"
attr(,"class")
[1] "unit_val"
```

```
x + y + z
```

```
[1] 307
attr(,"unit")
[1] "kg"
attr(,"class")
[1] "unit_val"
```

3. Métaprogrammation avec `substitute()`

`substitute()` capture une expression sans l'évaluer, permettant d'analyser ou de transformer le code.

```
log_call <- function(expr) {
    call_expr <- substitute(expr)
    result <- log(eval(expr, envir = parent.frame()))
    cat("Expression évaluée:", deparse(call_expr), "\nRésultat:", result,
"\n")
}

log_call(1 + 3)
```

```
Expression évaluée: 1 + 3
Résultat: 1.386294
```

4. Méthodes S3 et fonctions génériques

Le système S3 permet de définir des méthodes spécifiques pour des classes personnalisées.

```
individual <- function(name, age) {
    structure(list(name = name, age = age), class = "individual")
}

print.individual <- function(x, ...) {
    cat(sprintf("%s a %d ans\n", x$name, x$age))
}

p <- individual("Justine", 24)
print(p)
```

```
Justine a 24 ans
```

5. Arguments par défaut dépendants

La lazy evaluation permet de référencer d'autres arguments dans les valeurs par défaut.

```
interval <- function(
    mean,
    sd,
```

```
    confidence = 0.95,
    z_score = qnorm((1 + confidence) / 2),
    margin = z_score * sd
) {
    list(lower = mean - margin, upper = mean + margin, confidence =
confidence)
}

interval(mean = 100, sd = 15)
```

```
$lower
[1] 70.60054

$upper
[1] 129.3995

$confidence
[1] 0.95
```