RENCONTRES R 2026

Ce qui rend R unique

5 fonctionnalités qui *séduisent* un développeur Python 🐍

RÉSUMÉ

R possède des particularités qui peuvent surprendre quand on vient de Python, mais qui rendent le langage particulièrement puissant, notamment par sa **flexibilité**.

Ce poster présente **5 mécanismes** qui changent la manière d'écrire du code : des opérateurs lisibles, du dispatch léger, des expressions que l'on peut capturer, un système objet simple et des arguments évalués seulement quand ils sont utilisés.

LES 5 MÉCANISMES

01 Créer ses propres opérateurs

Nommer une intention directement dans la syntaxe.

R

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

5 %within% c(1, 10)    # TRUE
12 %within% c(1, 10)   # FALSE
```

Python

```python
class Infix:
    def __init__(self, f):
        self.f = f
    def __ror__(self, x):
        return Infix(lambda y: self.f(x, y))
    def __or__(self, y):
        return self.f(y)

within = Infix(lambda x, r: r[0] <= x <= r[1])

5 |within| (1, 10)   # True
12 |within| (1, 10)  # False
```

USAGE

Rendre le code plus proche du vocabulaire métier et du langage naturel.

COMMENT ÇA MARCHE

Un opérateur infixe est une fonction dont le nom est entouré par des %.

02 Surcharger des opérateurs

Faire agir un objet métier sur tout un vecteur.

R

```r
percent <- function(x) {
  structure(x / 100, class = 'percent')
}

`*.percent` <- function(x, prices) {
  unclass(x) * prices
}

percent(20) * c(100, 80, 50)
# 20 16 10
```

Python

```python
class Percent:
    def __init__(self, x):
        self.value = x / 100

    def __mul__(self, prices):
        return [self.value * p for p in prices]

Percent(20) * [100, 80, 50]
# [20.0, 16.0, 10.0]
```

USAGE

Exprimer « 20 % de ces prix » sans boucle ni appel technique.

COMMENT ÇA MARCHE

R appelle `*.percent()` puis applique naturellement le calcul au vecteur.

03 Capturer une expression

Traiter du code comme une donnée manipulable.

R

```r
label <- function(expr) {
  deparse(substitute(expr))
}

label(log(x + 1))
# log(x + 1)
```

Python

```python
# Pas possible en Python...
```

USAGE

Créer des messages, des formules, des messages de débogage ou des pipelines plus lisibles.

COMMENT ÇA MARCHE

`substitute()` récupère l'expression **avant** son évaluation. Python ne permet pas de passer une vraie expression en entrée.

04 Utiliser S3 simplement

Apprendre à `print()` comment afficher votre objet.

R

```r
order <- structure(
    list(id = 'A42', total = 89),
    class = 'order'
)

print.order <- function(x, ...) {
  cat(x$id, ':', x$total, 'EUR')
}

print(order)
# A42: 89 EUR
```

Python

```python
class Order:
    def __init__(self, id, total):
        self.id = id
        self.total = total

    def __str__(self):
        return f'{self.id}: {self.total} EUR'

print(Order('A42', 89))
# A42: 89 EUR
```

USAGE

Personnaliser l'affichage d'objets réels : commandes, modèles ou résultats.

COMMENT ÇA MARCHE

`print()` appelle `print.<classe>()` selon la classe de l'objet.

05 Profiter de l'évaluation paresseuse ("lazy")

Définir des valeurs par défaut qui dépendent des autres arguments.

R

```r
scale2 <- function(x, mu = mean(x), sigma = sd(x))
{
  (x - mu) / sigma
}
```

Python

```python
def scale2(x, mu = None, sigma = None):
    if mu is None:
        mu = mean(x)
    if sigma is None:
        sigma = sd(x)

    return (x - mu) / sigma
```

USAGE

Offrir des valeurs par défaut intelligentes sans demander plus à l'utilisateur.

COMMENT ÇA MARCHE

Les arguments sont des promesses : ils sont évalués seulement si nécessaire.

PDF EN LIGNE

SCAN ME

Joseph Barbier *Yellow Sunflower*
joseph@ysunflower.com