Opérateur pipe en R

Cette page est basé sur un document qu'Aurélien Nicosia (ULaval) a créé en 2023 appelé "Opérateur *pipe*". Celui-ci a été mis à jour.

Depuis la version 4.1.0 de R, sorti en mai 2021, le langage a introduit l'opérateur *pipe* |> en s'inspirant de ce que faisait le package magrittr. À partir de R 4.3.0, le guide de style du tidyverse recommande l'utilisation de l'opérateur *pipe* de base, et non celui de magrittr.

```
Raccourcis clavier

Dans RStudio, le raccourci clavier pour insérer l'opérateur pipe est :

— Sous Windows : Ctrl + Shift + M

— Sous macOS : + + M
```

Cet opérateur introduit une façon d'enchaîner les intructions et de passer des arguments à des fonctions de manière plus lisible que la manière classique.

Pour résumer le fonctionnement de cette opérateur, voici comment il transforme quelques appels de fonctions :

```
— f(x) devient x |> f();
— f(x, y) devient x |> f(y);
— h(g(f(x))) devient x |> f() |> g() |> h().
```

Cette opérateur permet de mettre en avant la **séquence d'actions** et non l'objet sur lequel la séquence d'actions est faite. Cela rend le code plus lisible (et avoir un code lisible est une bonne pratique). En lisant de gauche à droite l'instruction h(g(f(x))), nous voyons d'abord l'appel à la fonction h, puis l'appel à la fonction h get finalement l'appel à la focntion h. Pourtant, l'évaluation de cette instruction se fait dans le sens inverse. En effet, h va d'abord :

- 1. évaluer f(x);
- 2. puis, il passera le résultat à la fonction g et retournera le résultat;
- 3. qui sera passé à la fonction h et le résultat final sera retourné.

Si nous voulons écrire un code qui reflète l'ordre des évaluations correctement, nous pourrions écrire :

```
res1 <- f(x)
res2 <- g(res1)
h(res2)
```

Ce code a cependant le défaut de créer des objects que nous souhaitons pas nécessairement conserver. L'opérateur |> n'a pas ce défaut! En effet, une instruction écrite en utilisant l'opérateur |> permet de suivre l'ordre des évaluations, sans créer d'objets inutilement en mémoire.

Pour encore plus de clarté, il est possible d'étendre sur plusieurs lignes un instruction contenant plusieurs opérateur |> de façon à avoir une fonction par ligne :

```
x |>
f() |>
g() |>
h()
```

Si l'argument que nous souhaitons passer avec l'opérateur | > n'est pas celui en première position, il faut utiliser _ comme suit avec un paramètre nommé : f(y, z = x) devient $x | > f(y, z = _)$.

Prenons un exemple pour illustrer l'utilisation de l'opérateur $\mid >$. Supposons que nous avons la chaîne de caractères suivantes :

```
text <- "Ceci est un example"
```

et que nous souhaitons la corriger—remplacer "example" par "exemple" et ajouter un point à la fin—avec l'intruction suivante :

```
paste0(gsub(pattern = "example", replacement = "exemple", x = text), ".")
```

```
[1] "Ceci est un exemple."
```

Cette instruction est un peu difficile à lire en raison de l'appel à la fonction gsub imbriqué dans un appel de fontion paste0. Nous pourrions la réécrire comme suit avec l'opérateur |> :

```
text |>
  gsub(pattern = "example", replacement = "exemple", x = _) |>
  paste0(".")
```

[1] "Ceci est un exemple."

Prenons un autre exemple numérique. On souhaite faire le calcul suivant :

$$\frac{(2+4)\times 8}{2}.$$

Pour cela, nous avons besoin de quelques fonctions mathématiques.

```
add <- function(x, y) {
    x + y
}

mul <- function(x, y) {
    x * y
}

div <- function(x, y) {
    x / y
}</pre>
```

On peut faire le calcul de trois manières différentes :

```
# En créant différents objets
res1 <- add(2, 4)
res2 <- mul(res1, 8)
res3 <- div(res2, 2)
print(res3)</pre>
```

[1] 24

```
# En imbriquant les fonctions
res <- div(mul(add(2, 4), 8), 2)
print(res)</pre>
```

[1] 24

```
# Avec l'opérateur pipe
res <- 2 |>
  add(4) |>
  mul(8) |>
  div(2)
print(res)
```

[1] 24