Travaux dirigés pour l'introduction au logiciel R

Marco Pascucci

25/10/2018

Figure 1: The trachery of images - R.Magritte, oil on canvas, 1928–29

Figure 2: Sans titre - Moi, souris sur ordi, hier soir

Pipe = tuyau

c'est une **forme de syntaxe** qui enchaine deux fonctions/operations par entrée et sortie

Exemple:

$$y = \frac{1}{3}\sqrt{(2)^3 + 1}$$

On peux le lire comme:

divPar3 { racine[add1 (puissance3 (2))] }

ou bien:

2 —> puissance3 —> add1 —> racine —> divPar3

Pipe en R %>%

la syntaxe du PIPE en R est définie dans le package... `magrittr`

```
y <- (sqrt((2)^3+1))/3
```

est equivalent à

```
y_pipe <- 2 %>% "^"(3) %>% "+"(1) %>%sqrt() %>% "/"(3)
```

en fait:

```
y == y_pipe
```

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

PIPES avec des fonctions

On peut utiliser des fonctions dans un PIPE.

Si la fonction n'a qu'**un seul paramètre**, on peut utiliser juste son nom, sans ou avec parenthèses.

```
PI <- 3.14
# ces expressions donnent toutes le même resultat
sin(PI)
PI %>% sin
PI %>% sin()
```

Si la fonction a plus qu'un paramêtre, le PIPE passe au premier paramètre non utilisé... Pour plus de clarté, on utilise le signe-poste "." (POINT)

```
multiply <- function(a, b) { a * b }
v <- c(1,2,1,2,1,2,1)
reduce(v, multiply)
v %>% reduce(., multiply)
```

Un exemple

```r
compter <- function(quoi, ou) {
  # fonction qui compte combien de "quoi" il y a dans "ou"
  c = 0
  for (ca in ou) {
    if (ca == quoi){
      c = c+1
    }
  }
  c
}
v = c(0,0,1,0,1,0,1,0)

compter(1, v)

v %>% compter(1,.)
# v %>% compter(.,1) ne marchera pas... pourquoi?

v %>% compter(quoi=1,.) # et pourquoi ça marche maintenant?
v %>% compter(quoi=1)
```

Exercice 1

1. Implementer les fonctions suivantes:

- `add(a, b)` qui fait la somme de `a` et `b`
- `divPar(a,b)` qui divise `a` par `b`
- `puissance(a, exposant)` qui calcule `a` à la puissance `exposant` (donner à ce paramètre la valeur 1 par default)

2. En utilisant ces fonctions, écrire (sans et avec PIPES) l'instruction suivante, pour x = 2 :

$$y = \frac{1}{3}\sqrt{(x)^3 + 1}$$

3. Ecrire la fonction `doIt(x,a,b,c)` qui, avec des PIPES, calcule la formule suivante

$$\mathrm{doIt}(x, a, b, c) = \frac{1}{c}\sqrt{(x)^a + b}$$

Exercice 2 Utiliser la fonction `doIt()` pour trouver la valeur de x telle que `doIt(x,3,9,6) == 1`

Solution ex 1 et 2

```r
add <- function(a,b) { a+b }
divPar <- function(a,b) { a/b }
puissance <- function(a, exposant=1) { a^exposant }

# sans PIPES
y <- 1/3*(sqrt(2^3 + 1))
# avec PIPES
y <- 2 %>% puissance(exposant=3) %>% add(.,1) %>%
  sqrt() %>% divPar(.,3)

doIt <- function(x,a,b,c) {
  x %>% puissance(.,exposant=a) %>% add(.,b) %>%
    sqrt() %>% divPar(.,c)
}
```

```r
doIt(2,3,1,3) == 1   # TRUE
v <- 1:10
doIt(v,3,9,6)
```

Affectation à la volée

à la place de:

```
x <- x %>% add(.,10)
```

on peut utiliser l'operateur %<>% :

```
x %<>% add(.,10)
```

Exercice 3

Écrire ces affectation à la volée, partant d'une valeur x=2.

$$x = \frac{1}{3}\sqrt{(x)^3 + 1}$$

Exercice 3

Écrire cette affectation à la volée, en partant d'une valeur x=2.

$$x = \frac{1}{3}\sqrt{(x)^3 + 1}$$

Solution ex 3

```
x <- 2
x %<>% puissance(.,exposant=3) %>% add(.,1) %>%
  sqrt() %>% divPar(.,3)
```

T-PIPE

Certaine fonction n'ont pas de sortie, et on ne peut pas les utiliser dans un PIPE normal.

```
info <- function(x) {
  # imprime des informations sur le type et la valeur de x.
  print(paste("the parameter's type is:", typeof(x),
              "and it's value is:", x)
        )
}
2 %>% info()
```

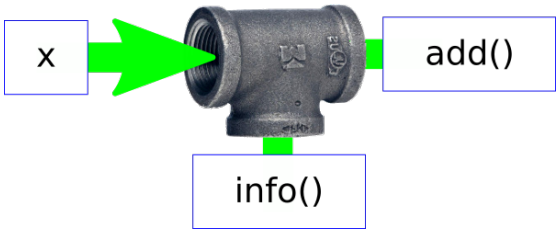
```
## [1] "the parameter's type is: double and it's value is: 2"
```

Donc ce PIPE là ne peut pas marcher (tester)

```
2 %>% info() %>% add(.,1) %>% info() # ne marche pas
```

T-PIPE

On utilise alors un T-PIPE.




```
2 %T>% info() %>% add(.,1) %>% info();
```

```
## [1] "the parameter's type is: double and it's value is: 2"
## [1] "the parameter's type is: double and it's value is: 3"
```

Exercice 4

Écrire un PIPE qui, à partir d'un array `x` de 101 valeurs -1 à 10:

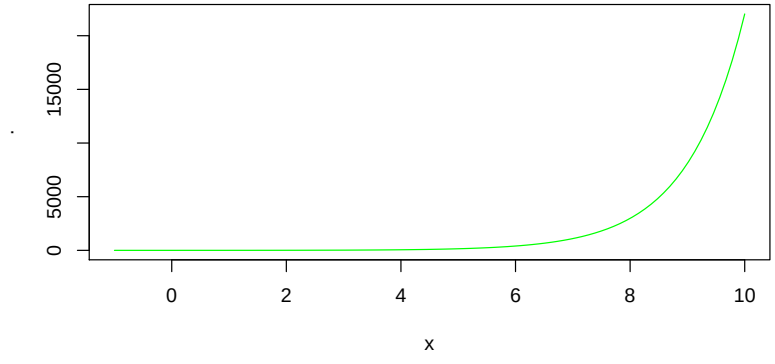
- calcule exp()
- fait le plot de exp() sur x
- calcule log(exp(x))
- fait le plot de ce dernier contre x
- calcule la moyenne de ce dernier

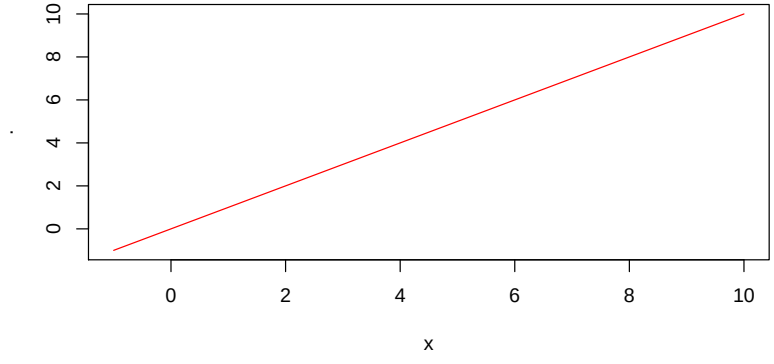
Solution ex. 4

```
# library(pracma)
# x <- linspace(-1,10,n=101)

x <- (0:101)*11/101 - 1

x %>% exp() %T>% plot(x,.,type='l', col='green') %>%
  log() %T>% plot(x,.,type='l', col='red') %>% mean(.)
```






```
## [1] 4.5
```