R avancé

partie 4 : Programmation fonctionnelle

romain@r-enthusiasts.com

Romain François

@romain_francois

Exercice: remplacer les -99 par des NA

```
> set.seed(42)
> df <- data.frame( replicate(6, sample(c(1:10,-99), 6, rep=TRUE )))
> names(df) <- letters[1:6]
> df
    a b   c   d  e  f
1 -99 9 -99   6  1  9
2 -99 2   3   7  6  9
3   4 8   6  10  5  5
4  10 8 -99   2 10  8
5   8 6 -99 -99  5  1
6   6 8   2 -99 10 10
```

```
df$a[ df$a == -99 ] <- NA
df$b[ df$b == -99 ] <- NA
df$c[ df$c == -98 ] <- NA
df$d[ df$d == -99 ] <- NA
df$e[ df$e == -99 ] <- NA
df$f[ df$e == -99 ] <- NA
```

```
fix_missing <- function(x){
  x[ x == -99 ] <- NA
  x
}
df$a <- fix_missing(df$a)
df$b <- fix_missing(df$b)
df$c <- fix_missing(df$c)
df$d <- fix_missing(df$d)
df$e <- fix_missing(df$e)
df$f <- fix_missing(df$f)
```

```
fix_missing <- function(x){
  x[ x == -99 ] <- NA
  x
}
df[] <- lapply( df, fix_missing )


missing_fixer <- function(na_value=-99){
  function(x){
    x[x==na_value] <- NA
    x
  }
}
df[] <- lapply( df, missing_fixer(-99) )
```

Fonctions anonymes

```
> formals( function(x=4) g(x) + h(x) )
$x
[1] 4

> body( function(x=4) g(x) + h(x) )
g(x) + h(x)
> environment( function(x=4) g(x) + h(x) )
<environment: R_GlobalEnv>
```

Fonctions anonymes

```
> (function(x) 3)()
[1] 3
> (function(x) x+3)(10)
[1] 13

> f <- function(x) x+3
> f
[1] 13
```

Closures
(fonctions retournées par des fonctions)

```
power <- function(exponent){
  function(x) x^exponent
}

square <- power(2)
cube <- power(3)

square(3)
cube(3)

> as.list(environment(cube))
$exponent
[1] 3
> as.list(environment(square))
$exponent
[1] 2
```

```r
l <- replicate( 20, runif(sample(1:10, 1)), simplify=FALSE )

# for
out <- vector("list", length(l))
for(i in seq_along(l)){
  out[[i]] <- length(l[[i]])
}
unlist(out)


# lapply
unlist(lapply(l, length))
```

Type de boucles for

- sur les éléments : for(x in xs){ ... }
- sur les indices: for(i in seq_along(xs)){ ... }
- sur les noms: for(nm in names(xs)){ ... }

```r
xs <- runif(1e3)

# SlooooooooW 🐌 🐌 🐌
res <- c()
for( x in xs ){
  res <- c(res, sqrt(x))
}

# un peu mieux 🐌
res <- numeric(length(xs))
for( i in seq_along(xs)){
  res[i] <- sqrt(xs[i])
}

# top:
res <- sqrt(xs)
```

```r
res <- c()
unlist(lapply(xs, sqrt))

lapply( seq_along(xs), function(i){
  sqrt(xs[i])
})
```

Arguments supplémentaires pour lapply (...)

```
> args(lapply)
function (X, FUN, ...)
NULL

lapply( l, mean, trim=0.2)

# pareil que
lapply( l, function(.){
  mean(., trim=0.2)
})
```

sapply / apply

```
sapply( l, mean, trim = 0.2 )
vapply( l, mean, numeric(1), trim = 0.2 )
```

```
xs <- replicate( 5, runif(10), simplify = FALSE )
ws <- replicate( 5, rpois(10,5), simplify = FALSE )
```

calculer :

moyennes de chaque element de xs

moyennes pondérées (weighted.mean) de chaque élément de xs avec l'élément correspondant de ws comme poids

```
xs <- replicate( 5, runif(10), simplify = FALSE )
ws <- replicate( 5, rpois(10,5), simplify = FALSE )

sapply(xs, mean)


sapply( seq_along(xs), function(i){
  weighted.mean( xs[[i]], ws[[i]] )
})
Map( weighted.mean, xs, ws )
mapply( weighted.mean, xs, ws )
```

```
> m <- matrix( 1:20, nrow = 5)

> apply(m, 1, mean)
[1]  8.5  9.5 10.5 11.5 12.5

> apply(m, 2, mean)
[1]  3  8 13 18
```