Introduction à R

Les types d'objets

Pascal Bessonneau

06/2015

Vecteurs

L'objet le plus courant est de loin le vecteur. il s'agit d'un tableau à 1 dimension stockant un seul et unique type de données.
La création d'un vecteur est soit implicite soit explicite.
Implicite avec l'opérateur *c* :

```
> c(2,3,4,6,7)

## [1] 2 3 4 6 7
```

Vecteurs

Soit explicite en appelant un créateur qui porte le nom du type que l'on souhaite stocké et sa longueur :

```
> numeric(4)

## [1] 0 0 0 0

> logical(4)

## [1] FALSE FALSE FALSE FALSE
```

Les fonctions associées aux types

Il existe deux familles de fonctions associées aux types :

is. les fonctions *is.* renvoie un booléen indiquant si la valeur appartient à un type donné

as. ces fonctions permettent de changer le type d'une valeur vers une autre (cast)

```
> is.integer(2L)

## [1] TRUE

> is.character(2.3)

## [1] FALSE

> as.character(2)

## [1] "2"
```

Matrices

Ensuite viennent les matrices qui sont des tableaux à deux dimensions :

```
> matrix( 0, ncol = 2, nrow = 2)

##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Les matrices sont créés avec un premier argument qui contient les données et une (ou deux tailles, largeur et/ou longueur).

Matrices

```
> matrix( 0, ncol = 2, nrow = 2)

##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Les matrices sont créés avec un premier argument qui contient les données et une (ou deux tailles, largeur et/ou longueur).
Le type de données, *unique*, stocké pour les matrices est le type de données par le premier argument.

les *data.frame*

Les *data.frames* sont des tableaux comme les matrices mais qui permettent de stocker des types de données différentes dans chaque colonne.

```
> head(data.frame(lettre=LETTERS,numero=1:26))

##   lettre numero
## 1      A      1
## 2      B      2
## 3      C      3
## 4      D      4
## 5      E      5
## 6      F      6
```

les *data.frame*

Les *data.frames* sont de loin la structure la plus utilisée en faisant de la manipulation de données.
Mais ce type dérive en fait d'un autre type de base moins manipulable par les débutants : les listes.

les *list*

Les *list* sont la structure la plus pratique. Ce sont des vecteurs où chaque élement du vecteur peut être un objet R quelconque y compris une liste.

```
> a=list(1, LETTERS, matrix(0,2,2))
> str(a)

## List of 3
##  $ : num 1
##  $ : chr [1:26] "A" "B" "C" "D" ...
##  $ : num [1:2, 1:2] 0 0 0 0
```

les *list*

Les *data.frames* sont en fait une liste avec comme condition que chaque élement soit un vecteur de même longueur (pour obtenir un tableau).
Ainsi les listes et les *data.frames* partagent beaucoup d'opérateurs en communs.

les *array*

les *array* sont des objets qui étendent les matrices à des tableaux à *k*-dimsions.

```
> array(1:4,c(2,2,2))

## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

les *list*

Les *data.frames* sont en fait une liste avec comme condition que chaque élement soit un vecteur de même longueur (pour obtenir un tableau).
Ainsi les listes et les *data.frames* partagent beaucoup d'opérateurs en communs.
Dans la programmation avancée, les *data.frames* sont souvent utilisées comme des listes.

les séries temporelles

Les séries temporelles sont des des élements pouvant stocker et avec des propriétés particulières des séries de date.
C'est le type utilisé pour tous les analyses en séries temporelles.

```
> ts(date())

## Time Series:
## Start = 1
## End = 1
## Frequency = 1
## [1] Sat May 21 04:29:45 2016
```

Les fonctions de transformation

Il existe deux familles de fonctions associées aux types :

- is. les fonctions *is.* renvoie un booléen indiquant si la valeur appartient à un objet donné
- as. ces fonctions permettent de changer un objet dans un autre type

```
> str(as.list(iris))

## List of 5
##  $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 .
##  $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
##  $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1

> str(as.matrix(iris[,1:4]))

##  num [1:150, 1:4] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
```

Dans cette partie...

... sont décrites les principales règles qui permettent de sélectionner une partie des objets les plus courants (le langage objet S4 est exclus).

Pour un vecteur

Pour un vecteur, l'opérateur d'extraction de valeurs sont des crochets. simples.
Comme on peut le voir sur les deux dernières lignes leur précédence n'est pas très forte mais plus grande que les opérateurs de calcul.

```
> LETTERS[1:10]

##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"

> c(1,2,3,4)[1:2]

## [1] 1 2

> c(1,2,3,4)[-4]

## [1] 1 2 3

> c(1,2,3,4)[-4] * 2

## [1] 2 4 6
```

Les trois aspects de l'indexation

L'indexation sauf certaine exception peut se faire avec trois types de données sous R.

entiers avec les entiers, les chiffres positifs indiquent les positions des vecteurs qui seront extraits. Les chiffres négatifs indiquent les positions à exclure. Attention on ne peux pas mélanger chiffre positif et chiffres négatifs

les entiers

Avec les entiers, les chiffres positifs indiquent les positions des vecteurs qui seront extraits. Les chiffres négatifs indiquent les positions à exclure.
La longueur est quelconque (mais supérieur ou égal à 1). La longueur corresponds au nombre de d'éléments à extraire (ou à ne pas extraire).
Attention on ne peux pas mélanger chiffres positifs et chiffres négatifs
Attention Attention, amoureux du C et du perl, le zéro n'est jamais un indice valide en R.

les entiers

La longueur du vecteur retourné est la longueur du vecteur de sélection pour les nombres d'entiers.
Pour les nombres négatifs, la longueur retournée est le total d'elements uniques du vecteurs moins la longueur du vecteur de sélection.

les entiers

```
> LETTERS[c(1,2,3,4)]

## [1] "A" "B" "C" "D"

> LETTERS[c(2,4)]

## [1] "B" "D"

> LETTERS[c(2,4,2)]

## [1] "B" "D" "B"
```

les entiers

```
> LETTERS[-c(1,2,3,4)]

##  [1] "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
## [11] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
## [21] "Y" "Z"

> LETTERS[-c(2,4)]

##  [1] "A" "C" "E" "F" "G" "H" "I" "J" "K" "L"
## [11] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V"
## [21] "W" "X" "Y" "Z"

> LETTERS[-c(2,4,2)]

##  [1] "A" "C" "E" "F" "G" "H" "I" "J" "K" "L"
## [11] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V"
## [21] "W" "X" "Y" "Z"
```

le vecteur logique

La grande différence avec les indices numériques est la longueur du vecteur.
Pour chaque position, si la valeur est *TRUE*, la valeur est retournée. Si c'est *FALSE*, la valeur n'est pas retourné.
Par conséquent la longueur du vecteur de sélection est la longueur du vecteur à séléctionner. Quant à la la longueur du vecteur de retour, c'est le nombre de *TRUE* (ie. la somme du vecteur logique).
Attention au recyclage ! Si le recyclage n'est pas possible R génère une erreur si la longueur des deux vecteurs ne coïncide pas.

le vecteur logique

```
> L <- LETTERS[1:6]
> L

## [1] "A" "B" "C" "D" "E" "F"

> L[c(T,F,T,F,T,T)]

## [1] "A" "C" "E" "F"

> L[c(T,F)] # recyclage

## [1] "A" "C" "E"
```

les noms

Les noms ont été évoqués brièvement. . . à peu près tout sous R peut porter un noms.
On peut donc utiliser un vecteur *character* avec le noms des élements pour les récupérer.
Les noms peuvent être utilisés en lieu et place des numéros. La longueur du vecteur de retour est alors le nombre de noms mis en arguments.

les noms

```
> (a <- 1:4)

## [1] 1 2 3 4

> (names(a) <- LETTERS[1:4])

## [1] "A" "B" "C" "D"

> a[c("A","D","A","C")]

## A D A C
## 1 4 1 3
```

Généralisation du système d'indexation

Une fois compris ce système d'indexation, le plus dur est fait car c'est sur ce système que se base pratiquement tout l'indexation des lignes et des colonnes d'une matrice, des élements d'une liste, ... Quand l'objet est composé de lignes et de colonnes il suffit d'indexer et d'ajouter une "," pour indiquer à R sur quel dimensions on travaille.

data.frames et matrices

```
> str(iris[c(1,3,4),]) # à gauche -> lignes

## 'data.frame': 3 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.7 4.6
##  $ Sepal.Width : num  3.5 3.2 3.1
##  $ Petal.Length: num  1.4 1.3 1.5
##  $ Petal.Width : num  0.2 0.2 0.2
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1

> str(iris[,c(2,4,5)]) # à doirte -> colonnes

## 'data.frame': 150 obs. of  3 variables:
##  $ Sepal.Width: num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Width: num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species    : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1

> str(iris[c(2,4,5),c(2,4,5)]) # les deux

## 'data.frame': 3 obs. of  3 variables:
##  $ Sepal.Width: num  3 3.1 3.6
```

Listes

Les listes sont proches des vecteurs. Les mêmes règles peuvent être appliquées.
Il y a toutefois une subtilité. Entre crochets simples, les listes renvoie une liste. Mais entre crochets doubles, un seul élement peut être renvoyé mais l'élement n'est pas de type liste mais du type contenu dans la liste à cette position.
C'est logique puisque vu l'hétérogénéité des élements pouvant être stockés dans une liste, R ne peut déterminer la meilleure stratégie pour rendre une série d'objets hétérogène.
Par contre quand un seul objet est renvoyé ce problème ne se pose pas.

Listes

```
> a <- list(1, LETTERS, matrix(0,2,2))
> a[c(T,F,T)]

## [[1]]
## [1] 1
##
## [[2]]
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0

> a[[1]]

## [1] 1
```