Introduction au langage R

Algo & Prog avec R

A. Malapert, B. Martin, M. Pelleau, et J.-P. Roy

3 avril 2019

Université Côte d'Azur, CNRS, I3S, France
firstname.lastname@univ-cotedazur.fr

Ressources sur cet enseignement

La page Web du cours, que vous enregistrez dans vos signets.

`www.i3s.unice.fr/~malapert/R`

Langage de programmation

Environnement de développement

Qu'est-ce que la programmation ?

L'art de rédiger des textes dans une langue artificielle.

Art Science ?

Textes Programmes

Langue artificielle Langage de programmation

Dans quel but ?
Faire exécuter une tâche à un ordinateur.

Il s'agit d'une activité de résolution de problèmes.

- Hum, on va faire des maths, alors ?
- Un peu quand même, mais pas trop, le monde est vaste. Nous allons tâcher d'en modéliser de petites portions pour les faire rentrer dans la machine. Des nombres, du texte, des images, le Web ...
- On pourrait presque dire : calculer le Monde ?
- Oui, bravo, c'est cela, réduire le Monde à des ensembles d'objets sur lesquels on peut faire des calculs.

Jouer avec des nombres entiers

Dans toutes les sciences, les nombres jouent un rôle important.
R se présente comme une calculette interactive à travers son toplevel. Il présente son prompt > pour que vous lui soumettiez un calcul ...

```
> 10 + 3
[1] 13
> 10 + 3 * 5
[1] 25
> 16 / 3
[1] 5.333333
> 16 %/% 3
[1] 5
> 16 %% 3
[1] 1
```

Opérateurs de base sur les entiers

- + et − : addition et soustraction.
- ∗ et / : multiplication et division.
- ∗∗ : puissance.
- %/% ou %% : quotient et reste.

Le kilo informatique

```
> 2**10 # le kilo informatique
[1] 1024
```

Division euclidienne

Si a et b sont deux entiers avec $b \neq 0$, alors :

$$a = b * (a\%/\%b) + (a\%\%b)$$

- a%/%b fournit le quotient de la division de l'entier a par b.
- a%%b fournit le reste de la division de l'entier a par b.

```
> 16 %/% 3
[1] 5
> 16 %% 3
[1] 1
> 3 * (16 %/% 3) + (16 %% 3)
[1] 16
```

Erreurs de calcul

Mal utilisée, une opération peut provoquer une ERREUR.

Gestion des erreurs avec une valeur spéciale

```
> 16 %\% 0
[1] NaN
> 0/0
[1] NaN
```

Gestion des erreurs avec une exception

```
> 'a' + 5
Error in "a" + 5 : argument non numérique pour un opérateur
    binaire
```

Priorité des opérations

L'ordre du calcul d'une expression arithmétique tient compte de la priorité de chaque opérateur.

Priorité	Opérateurs
Faible	+ et -
Haute	*, /, %/% et %%
Très haute	**

```
> 5 - 8 + 4 * 2 ** 3
[1] 29
> (5 - 8) + (4 * (2 ** 3))
[1] 29
```

Dans le doute, mieux vaut mettre des parenthèses !

Représentation des nombres (voir l'autoformation)

La taille des entiers est toujours limitée par la mémoire de la machine.

En R, les entiers sont représentés sur 32 bits.

```
> .Machine$integer.max
[1] 2147483647
```

On dit que l'on travaille avec des nombres entiers en précision finie !

Pourtant, R peut renvoyer un résultat entier au-delà de cette valeur.

```
> 2^31
[1] 2147483648
```

R va automatiquement transformer le résultat en un nombre flottant. Les nombres flottants représentent les nombres réels sur 64 bits. Attention, l'arithmétique flottante n'est pas exacte !

```
> x <- 2 ** 221
> x + 1 == x
[1] TRUE
```

Comment expliquer vous ce résultat ? Quels sont les autres problèmes ?

Prgrammer avec des variables

Variable

Un peu comme les inconnues en Algèbre, sauf qu'ici une variable devra être connue et contenir une valeur.

```
> a <- 2 # lire : a prend pour valeur 2
> p <- 10 # p prend pour valeur 10
> c <- a ** p # c prend pour valeur celle de a p
> c  # toujours le kilo informatique
[1] 1024
```

Instruction d'affectation : `variable <- expression`

Ce signe <- (ou =) non commutatif n'a rien à voir avec celui des maths !

L'écriture `2 + 3 = a` n'aura donc aucun sens ! !

```
> 2 + 3 = a
Error in 2 + 3 = a :
  la cible de l'assignation␣est␣un␣objet␣n'appartenant pas
      au langage
> 2 + 3 = 5
Error in 2 + 3 = 5 : ...
```

Expression ou instruction ?

Ne confondez pas les EXPRESSIONS et les INSTRUCTIONS, qui toutes deux vont contenir des variables.

- Une expression sera calculée,
- tandis qu'une instruction sera exécutée !

```
> a <- 5
> x <- 3
```

Expression
Renvoie un résultat.

```
> a * x ** 2
[1] 45
```

Instruction
Aucun résultat.

```
> c <- a * x ** 2
>
```

les séparateur d'instructions : saut de ligne et le point virgule.

```
> a <- 5 ; x <- 3
```

```
  test
```

Une variable a le droit de changer de valeur

```
> a <- 2
> b <- a # la valeur de a est calculée puis c'est elle qui
    est transmise à b
> a <- a + 1 # qui se lit : a "devient égal à " a + 1
> b # et non 3, ok ?
[1] 2
```

Échange de deux variables
Avec une variable temporaire.

```
> temp <- a
> a <- b
> b <- temp
> a
[1] 2
> b
[1] 3
```

Comparaison

Opérateurs de comparaison

```
> a <- 2
> a == 3 # égalité mathématique
[1] FALSE
> a > 3 # strictement supérieur
[1] FALSE
> a + 1 >= 3 # supérieur ou égal
[1] TRUE
> a + 1 < 3 # strictement inférieur
[1] FALSE
> a + 1 <= 3 # inférieur ou égal
[1] TRUE
> a + 1 != 3 # différent
[1] FALSE
```

Booléens `TRUE` et `FALSE`

Dans une expression arithmétique,

- `TRUE == 1`,
- `FALSE == 0`.

```
> 5 + TRUE
[1] 6
> TRUE * FALSE
[1] 0
> FALSE + 1
[1] 1
> FALSE + 1 == TRUE
[1] TRUE
```

Evitez ce genre de facilités !

Affichage de résultats avec `printf`

La fonction print(...) permet d'afficher une expression :

```
> a = 2
> print(a * a)
4
>
```

Ce qui est affiché n'est pas le résultat de la fonction `print`, mais l'effet de cette fonction.

Cependant, la fonction renvoie aussi l'objet `x` passé en paramètre de manière invisible.

```
> print(5)==5
[1] 5                    # l'effet de print(5)
[1] TRUE                 # le résultat du test d'égalité
>
```

Construction de résultats avec `paste`

La fonction `paste(...)` permet de concaténer une expression :

```
> paste('le carre de', a, 'vaut', a*a)
[1] "le carre de 2 vaut 4"
```

Remarquez l'espace automatique.

- Ce qui est affiché est le résultat de la fonction `paste`.
- La fonction `paste` accepte des paramètres optionnels modifiant le format du résultat.

```
> paste('le carre de', a, 'vaut', a*a, sep='|')
[1] "le carre de|2|vaut|4"
```

Cependant, les possibilités de la fonction paste(...) sont limitées.

Construction de résultats avec `sprintf`

La fonction `sprintf(...)` permet de concaténer et formater finement une expression :

```
> sprintf('le carre de %d vaut %d', a, a*a)
[1] "le carre de 2 vaut 4"
```

- Ce qui est affiché est le résultat de la fonction sprintf.
- Les fonctions de la famille `printf` sont très puissantes et disponibles dans de nombreux langages.

```
> sprintf('le carre de %.5f vaut %08.5f', pi, pi*pi)
[1] "le carre de 3.14159 vaut 09.86960"
```

Rédaction de texte dans un éditeur

Le travail interactif au toplevel est pratique pour de petits calculs. Mais mieux vaut en général travailler dans un éditeur de texte.

Conditionnelle : la prise de décisions `if`

L'instruction conditionnelle if permet de prendre une décision.

Dans l'éditeur

```
a <- -2
if (a > 0) {
  paste(a,'est␣positif')
} else {
  paste(a,'est␣negatif␣ou␣nul')
}
```

RUN → -2 est négatif ou nul

Au toplevel

C'est moins agréable, il faut faire attention au(x) saut(s) de ligne.

```
> a <- -2
> if(a>0) {paste(a,'est␣positif') }
> else {paste(a,'est␣negatif␣ou␣nul')}
```

RUN → Erreur : 'else' inattendu(e) in "else"

Bloc et indentation

Indentation

La bonne distance à la marge d'une ligne permet de structurer et de comprendre un programme.

```
a <- -2
if (a > 0) {
  paste(a,'est positif')
} else {
  paste(a,'est negatif ou nul')
}
```

Bloc d'instructions

Un bloc d'instructions est une suite d'instructions alignées à la verticale.

Il est nécessaire d'ouvrir et fermer les paires d'accolades : $\{\ldots\}$.

```
a <- -2
if (a > 0) {
  paste(a,'est positif')
else {
  paste(a,'est negatif ou nul')
}
```

RUN ⟶ Erreur : 'else' inattendu(e) in "else"

Opérateurs logiques : ET (&&) et OU (||)

Table de vérité : ces opérateurs ressemblent à ceux de la Logique.

p	TRUE	TRUE	FALSE	FALSE
q	TRUE	FALSE	TRUE	FALSE
p && q	TRUE	FALSE	FALSE	FALSE
p || q	TRUE	TRUE	TRUE	FALSE

Mais, ils sont court-circuités

```
> a <- -2
> x == 3
Erreur : objet 'x' introuvable
> (a > 0) && (x==3)
[1] FALSE
```

L'expression `x == 3` n'a pas été évaluée car `FALSE && ? == FALSE`

La priorité de && étant plus faible que celle des opérations arithmétiques, on aurait pu écrire : `a > 0 and x == 3 .`

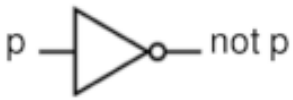
Liens avec l'électronique numérique

L'opérateur || est aussi court-circuité

```
> a <- -2
> x == 3
Erreur : objet 'x'
    introuvable
> (a < 0) || (x==3)
[1] TRUE
```

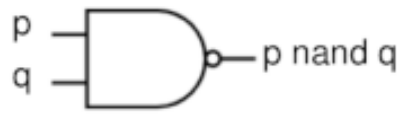
car `TRUE || ? == TRUE`

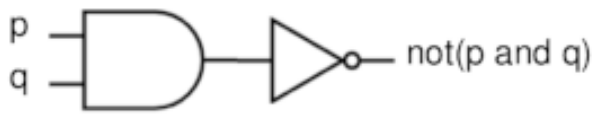
Portes logiques en électronique

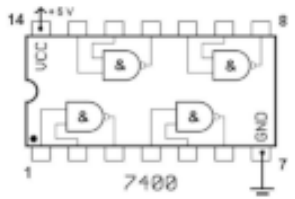


L'opérateur ! inverse les valeurs `TRUE` et `FALSE`. C'est l'inverseur ...

Les constructeurs d'ordinateurs utilisent beaucoup la porte nand.







Les fonctions prédéfinies de R

Tous les langages de programmation fournissent un large ensemble de fonctions prêtes à être utilisées.

Exemples dans les entiers

```
> abs(-5) # la fonction "valeur absolue"
5
> '+'(4,9) # les opérateurs sont des fonctions cachées
13
```

Certaines fonctions résident dans des modules/packages spécialisés, comme `TurtleGraphics` ou `shiny` ...

```
> turtle_forward(dist=15) # je veux réaliser un graphisme
    tortue
Erreur : impossible de trouver la fonction "turtle_forward"
> library(TurtleGraphics) # il faut charger l'extension
> turtle_init()
> turtle_forward(dist=15)
```

Comment définir une nouvelle fonction ?

Syntaxte pour la définition d'une fonction

```
nomFonction <- function(listeDeParamètres) {
  blocInstructions
  return(résultatFonction)
}
```

Le mot return signifie "le résultat est ... ".

Définir la fonction $f(n) \to 2n+1$

```
> f <- function(n) {return( 2*n - 1)}
> f(5)
[1] 9
```

Paramètres non typés : `n` n'est pas forcément un entier.

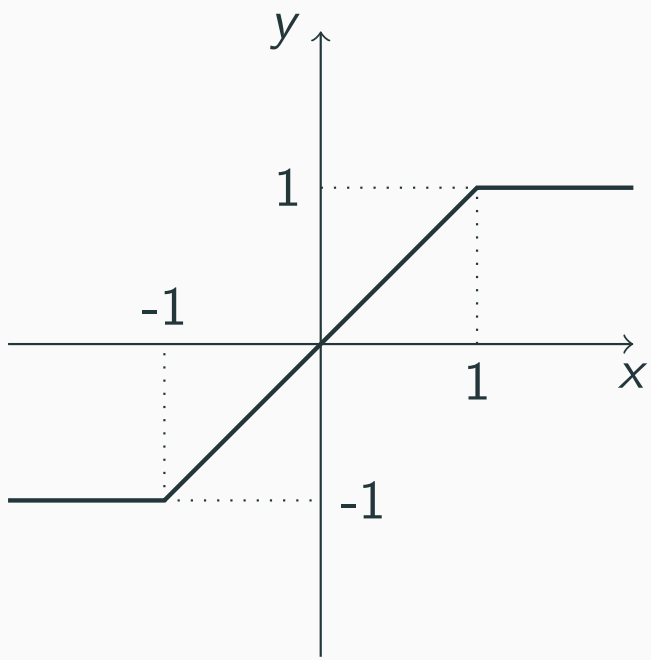
```
> f(5.2) # avec des réels approchés
9.4
> f(complex(real = 1, imaginary = 1)) # avec des complexes
[1] 1+2i
```

Les choix multiples avec `if ... else if ... else ...`

```
f <- function(x) {
  if( x < -1) return(-1)
  else if( x > 1) return(1)
  else return(x)
}
```

ou de manière équivalente

```
f <- function(x) {
  if( x < -1) return(-1)
  else {
    if( x > 1) return(1)
    else return(x)
}
```



Exemples

Comment (re)définir la valeur absolue ?

```
Abs <- function(n) {
if(n>0) {
  return(n)
} else {
  return (-n)
}
```

```
> paste('|-5|␣vaut', Abs(-5))
[1] "|-5|␣vaut␣5"
```

Comment (re)définir le maximum ?

```
Max <- function(n, m)
    {
if(n > m) {
  return(n)
} else {
  return (m)
}
```

```
> paste('max(-10,␣-5)␣vaut', Max(-10,
    -5))
[1] "max(-10,␣-5)␣vaut␣5"
```

Notions élémentaires sur les nombres premiers

Nombre premier
Un nombre premier ne peut être divisé que par lui-même et par 1.

Plus grand commun diviseur (PGCD)
Le PGCD de deux nombres entiers non nuls est le plus grand entier qui les divise simultanément.

Premiers entre eux
on dit que deux entiers sont premiers entre eux si leur plus grand commun diviseur est égal à 1.

Supposons que la fonction `gcd(p, q)` existe et renvoie le plus grand diviseur commun des entiers `p` et `q` (nous la programmerons en TP).

Composition de fonctions I

Créons une fonction `premiersEntreEux(p,q)` basée sur la fonction `gcd`.

```
premiersEntreEux <- function(p,q) {
  if (gcd(p,q) == 1) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}
```

ou encore

```
premiersEntreEux <- function (p,q) {
  if (gcd(p,q) == 1) {
    return(TRUE)
  }
  return(FALSE)
}
```

Le mot clé `return` provoque un échappement. Le reste du texte de la fonction est abandonné !

Composition de fonctions II

Une version qui renvoie directement le résultat de l'évaluation de l'expression.

```
premiersEntreEux <- function(p,q) {
  return(gcd(p,q) == 1)
}
```

Une version encore plus courte qui omet les accolades et `return`.

```
premiersEntreEux <- function(p,q) gcd(p,q) == 1
```

```
> premiersEntreEux(21,6)
[1] FALSE
> premiersEntreEux(21,8)
[1] TRUE
```

Vous voyez qu'il existe différentes manières de coder une fonction. Elles se distinguent par leur efficacité, mais aussi leur élégance.

Documenter une fonction

Pour l'instant
Ajouter simplement des commentaires au début de la fonction.

```
premiersEntreEux <- function(p,q) {
  # Détermine si deux entiers p et q sont premiers entre eux
      par calcul du PGCD.
  #
  # Arguments:
  #  p un entier
  #  q un entier
  #
  # Returns: TRUE si p et q sont premiers entre eux, et
     FALSE sinon.
  ...
}
```

En effet, il suffit de taper le nom d'une fonction pour voir son code.

Consulter la documentation de R

```
> ?paste # consulter la documentation d'une fonction
Concatenate Strings

Description:

     Concatenate vectors after converting to character.

Usage:

     paste (..., sep = "␣", collapse = NULL)
     paste0(..., collapse = NULL)

Arguments:
  ...
> ??paste # rechercher dans la documentation
```

Questions?

Retrouvez ce cours sur le site web

`www.i3s.unice.fr/~malapert/R`