

# Expressions, instructions, et variables

## Algo & Prog avec R

---

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

16 juin 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 calculatrice 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
```

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 ?



# Programmer 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** !

## Expression

Renvoie un résultat.

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

## Instruction

Aucun résultat.

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

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

**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
```

## 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
```

Évitez ce genre de facilités !

# Affichage de résultats avec print

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"
```

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





Questions?

Retrouvez ce cours sur le site web

[`www.i3s.unice.fr/~malapert/R`](http://www.i3s.unice.fr/~malapert/R)

# Test de la coloration syntaxique

```
stopifnot "stopifnot" # "stopifnot"  
MyFunc "MyFunc" # MyFunc  
while "while" # while  
runif "runif" # runif  
1 + 2.5 > 0 "1 + 2.5 > 0" # 1 + 2.5 > 0
```