Conditions et fonctions

Algo & Prog avec R

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

7 octobre 2019

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

Conditionnelle : la prise de décisions `if`

L'instruction conditionnelle `if` permet de prendre une décision.

Dans l'éditeur

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

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) {cat(a,'est positif\n') }
> else {cat(a,'est negatif ou nul\n')}
```

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

Bloc et indentation

Syntaxe pour une instruction conditionelle

```
if (condition) {
  blocInstructions
} else {
  blocInstructions
}
```

Indentation

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

Bloc d'instructions

C'est une suite d'instructions délimitée par des accolades.

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

Attention aux parenthèses et accolades !

Une utilisation correcte des accolades est obligatoire en R
Il est nécessaire d'ouvrir et fermer les paires d'accolades : $\{\ldots\}$.

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

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

Une utilisation correcte des parenthèses est obligatoire en R
Une paire de parenthèses doit toujours délimiter la condition du `if` : $(\ldots)$.

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

RUN → Erreur : unexpected symbol in "if a"

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 && x == 3` .

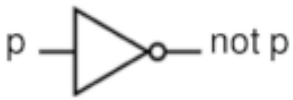
Liens avec l'électronique numérique

L'opérateur || est aussi court-circuité

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

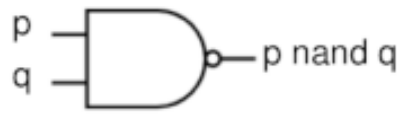
car `TRUE || ? == TRUE`

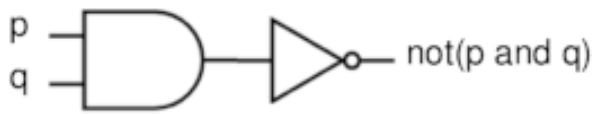
Portes logiques en électronique

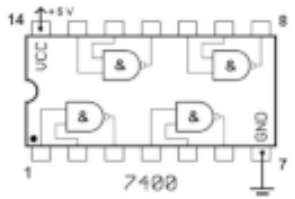


L'opérateur ! inverse les valeurs `TRUE` et `FALSE`. C'est l'inverseur ...

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 ?

Syntaxe pour la définition d'une fonction

Une fonction est une routine qui retourne une valeur.

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

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

Définir la fonction $f(n) \rightarrow 2n - 1$

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

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

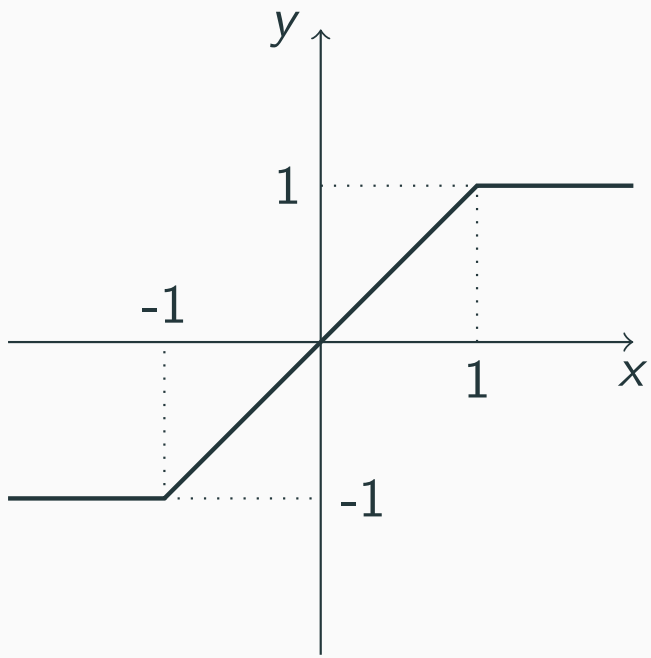
```
> f(5.2) # avec des réels approchés
9.4
> f(complex(real = 1, imaginary = 1)) # avec des complexes
[1] 1+2i
```

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)
  }
}
```

```
> cat('|-5| vaut', Abs(-5), '\n')
|-5| vaut 5
```

Comment (re)définir le maximum ?

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

```
> cat('max(-10, -5) vaut', Max(-10,
    -5), '\n')
max(-10, -5) vaut -5
```

Exemple : année bissextile

Depuis l'ajustement du calendrier grégorien, l'année sera bissextile (elle aura 366 jours, et non 365) :

- si l'année est divisible par 4 et non divisible par 100, ou
- si l'année est divisible par 400.

Comment calculer le nombre de jours d'une année ?

```
JoursParAn <- function(n) {
  if(((n %% 4 == 0) && (n %% 100 != 0)) || (n %% 400 == 0)){
    return(366)
  } else {
    return(365)
  }
}
```

Remarquez l'utilisation correcte des parenthèses.

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`