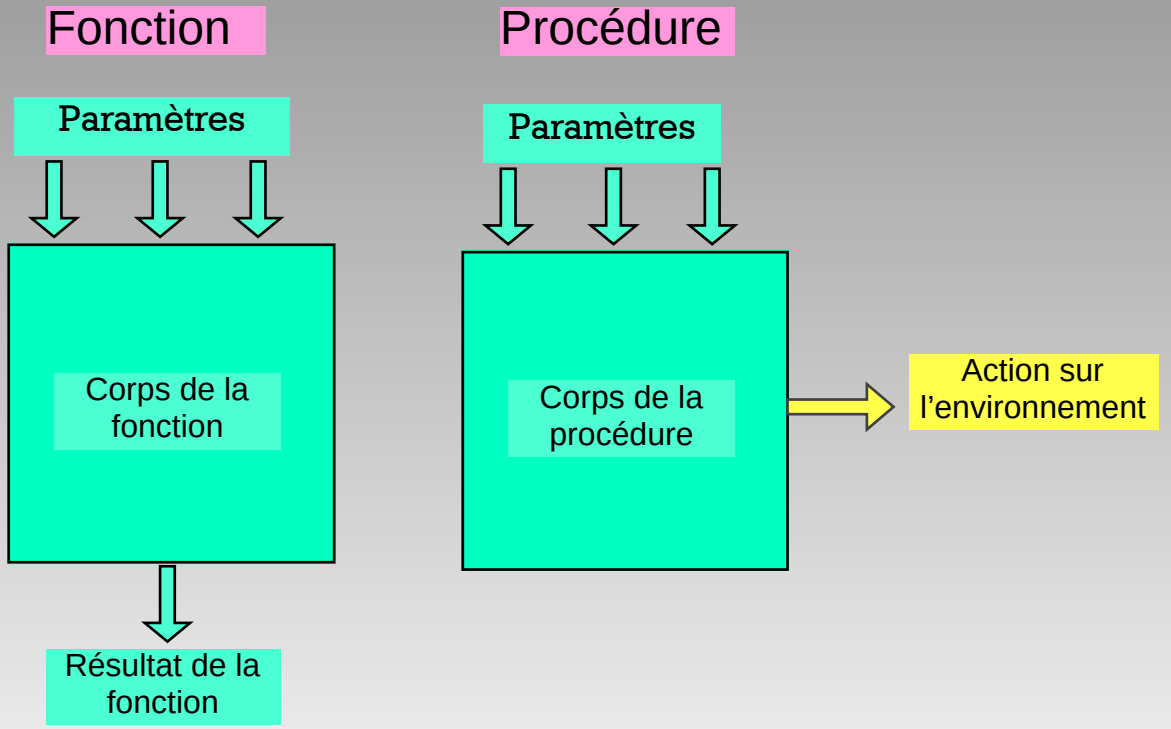
Programmation de Maple

par Denis Monasse

Lycée Louis le Grand, Paris

- **Utilisation libre pour usage pédagogique**

Fonctions et procédures



Passage de parametres

$f: x \mapsto \sin(2x+3)$ x est paramètre formel de f

$f(\frac{3\pi}{2}+t)$ $\frac{3\pi}{2}+t$ est paramètre effectif de f

Règle

- **Avant de procéder au calcul, Maple évalue les paramètres effectifs puis remplace dans le calcul toutes les occurences des paramètres formels par les valeurs des paramètres effectifs correspondants**

Structure générale d'une fonction ou procédure

proc(*paramètres formels*)

local *noms des variables locales* ;

instruction;

instruction;

.....

instruction

end

Nommer une fonction ou procédure

```
f := proc(....)
   local ....;
      ................
      ................
   end;
```

Parametres

Paramètres non nommés

- nombre de paramètres : `nargs`
- valeurs des paramètres: `args[1],…,args[nargs]`
  - » exemple : `f(18,t+3)`
    - nargs vaut 2, args[1] vaut 18, args[2] vaut t+3

Paramètres nommés

- On donne un nom à chaque paramètre formel; les valeurs attribuées à ces noms (les paramètres effectifs) peuvent alors être utilisées à l'intérieur de la procédure à l'aide de ces noms.
  - » Exemple : `f:= proc(x,y1,toto) ..... end;`

Sortie et retour d'un resultat

L'évaluation de l'expression Maple

RETURN(*expression*)

à l'intérieur du corps d'une fonction provoque l'évaluation de *expression*, la sortie immédiate du corps de la fonction avec comme résultat la **valeur** de *expression*.

En l'absence d'une telle instruction, la fonction retourne le résultat (éventuellement *rien*) de la dernière instruction effectuée dans le déroulement de l'exécution du corps de la fonction.

Variables locales

Les variables locales sont utilisées pour stocker temporairement à l'intérieur de la fonction des résultats intermédiaires; elles sont nommées à la suite du mot clé `local` et avant toute instruction.

Les noms de ces variables locales n'interfèrent pas avec ceux des variables globales: ils se contentent de masquer **temporairement**, pendant l'exécution du corps de la fonction, des variables globales de même nom.

Le contenu de ces variables est **détruit** à la sortie de la fonction

Instructions elementaires

Affectation

- Permet de stocker un résultat dans une variable globale ou locale
- Sous la forme *nom* := *expression*. Lors de l'évaluation de cette affectation, *expression* est tout d'abord évaluée et sa valeur est affectée à *nom*. A partir de cet instant, toute évaluation de *nom* renverra cette valeur comme résultat.

Effet de bord

- Appel de procédure avec des paramètres effectifs pour effectuer une action sur l'environnement

Séquence d'instructions

- Suite d'instructions élémentaires ou composées séparées par des points virgules; les résultats eventuels d'appels de fonctions sont ignorés (sauf pour la dernière instruction du corps de la fonction qui sert éventuellement de valeur de retour)
  - » Exemple :

```
-x:= sin(3*t);
-y:= 2*x + Pi/4;
-print(x+1);
-sin(y);  # inutile, ce résultat est ignoré
-z:= x+y
```

INSTRUCTION CONDITIONNELLE

Si une condition est vérifiée, on fait quelque chose, sinon on fait autre chose.

```
if condition then séquence d'instructions
[else autre séquence d'instructions]
fi
```

La partie entre crochets est facultative.
L'évaluation de *ccondition* doit nécessairement renvoyer un résultat de type booléen, vrai ou faux (`true` ou `false`).

Dans des instructions conditionnelles emboîtées (si une condition est vérifiée on fait quelque chose, sinon, si une deuxième condition est vérifiée on fait une deuxième chose, sinon ...), on peut condenser `else if` en `elif` ce qui donne la structure générale

```
if    condition 1 then séquence d'instructions 1
elif  condition 2 then séquence d'instructions 2
.....................................
elif  condition N then séquence d'instructions N
else  autre séquence d'instructions
fi
```

INSTRUCTION ITERATIVE

On répète une même séquence d'instructions en faisant varier un compteur jusqu'à ce que ce compteur dépasse une certaine valeur ou qu'une condition ne soit plus vérifiée.

```
[for compteur ][from début][to fin][by pas][while condition]
        do séquence d'instructions od
```

• la variable de nom *compteur* est initialisée à la valeur de *début*,

• la séquence d'instructions est répétée en donnant au compteur les valeurs successives *début*, *début*+*pas*, *début*+2*pas*, . . .

• jusqu'à ce que

- soit la condition ne soit plus vérifiée,
- soit le compteur dépasse strictement la valeur de *fin*.

Les tests sont effectués avant l'exécution de la séquence d'instructions (celle-ci peut donc très bien en jamais être effectuée).

Toutes les parties entre crochet sont facultatives, les défauts étant :

• le compteur n'a pas de nom, le début vaut 1, la fin vaut l'infini, le pas est de 1, la condition est toujours vraie.

Plutôt que de faire croître le compteur, on peut le faire décroître en remplaçant le mot `to` par `downto`.

Types de données

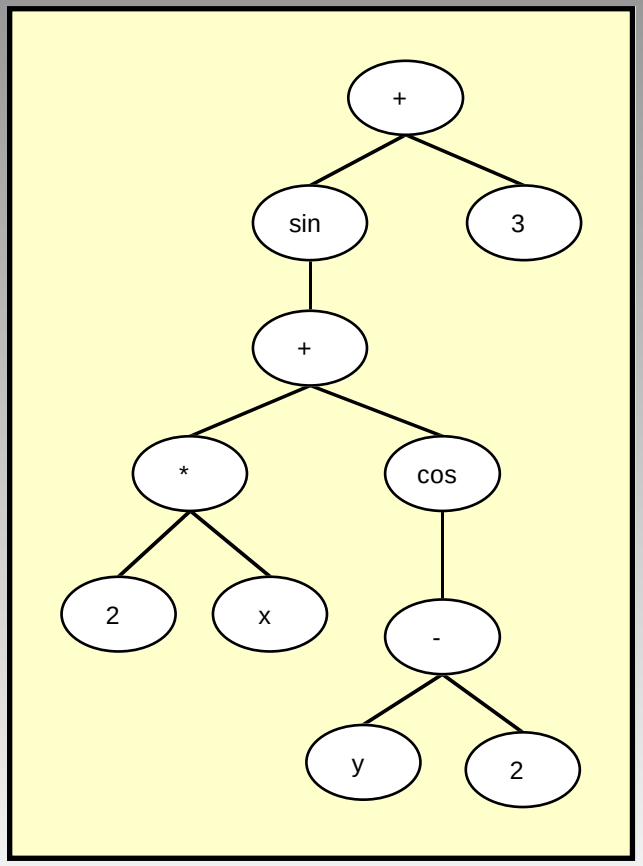
- nombre entier ou `integer` : 1234567891458965 7
- nombre réel ou `float` : 2.71828
- chaine de caractères ou symbole : `x , toto , sin , Pi`
- booléen : `true, false`
- séquence : *expression1* , ..., *expressionN*
- famille mathématique ou `list` : [*séquence*]
- ensemble mathématique ou `set` : { *séquence* }
- tables ou matrices : voir la documentation de `array` ou de `matrix`

Grammaire des expressions

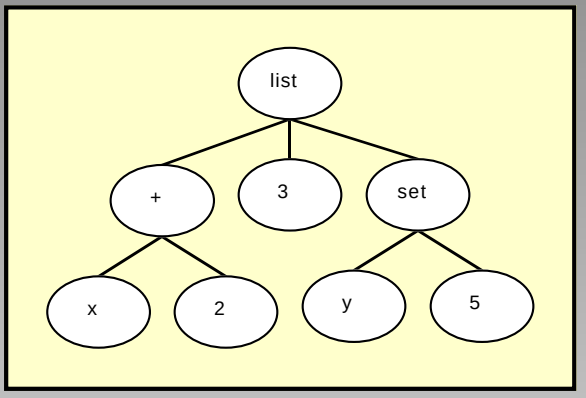
- operateur ::= `+` | `-` | `*` | `/` | `^` | `&*` | `etc`
- expression elementaire ::= entier
- | chaine de caractères
- expression ::= expression elementaire
- | expression elementaire `operateur` expression
- | expression(expression , … , expression)

Arbre d'une expression

$\sin(2*x+\cos(y-2))+3$



[x+2,3,{y,5}]



Les fonctions op et nops