Mathématica Cours 9

Le texte écrit en police `Typewriter` correspond au langage mathématica

Programmation récursive

L'exemple de la factorielle

Avec Mathematica, il existe souvent de nombreuses façons de programmer une même fonction. Voici par exemple cinq façons très différentes de définir une fonction f calculant la factoriel d'un entier naturel

1. `f[n_]:=n!`
N'est que l'utilisation d'une fonction déjà existante

2. `f[n_]:=Product[k,{k,1,n}]`
N'est que la traduction de la formule mathématique et ne met pas en jeu de réelle connaissance de programmation

3. `f[n_]:=Module[{p=1},Do[p=p*k,{k,2,n}];p]`
Voilà une programmation procédurale où toutes les opérations à effectuer sont explicitées étape par étape. C'est un style de programmation très classique

4. `f[n_]:=If[n==0,1,n*f[n-1]]` Est en apparence paradoxal puisque pour définir f, on l'utilise comme si elle exisatit déjà ! Cela s'appelle programmation récursive car traduit tout simplement une formule de récurrence mathématique, avec sa condition initiale. Bien noter qu'une programmation récursive commence toujours par une condition initiale, appelée plutôt test d'arrêt en informatique

5. `f[0]=1;f[n_]:=n*f[n-1]`
C'est une variante de la quatrième définition

1. Saisir ces fonctions en testant à chaque fois sur quelques exemples

Deux exemples

2. Programmer une fonction pm à deux arguments permettant le calcul de la puissance montante d'ordre $n \geq 1$ d'un réel a : $a(a+1)\cdots(a+n-1)$

3. Programmer le calcul du coefficient binomial $\binom{n}{k}$ de façon récursive

Limitation de la programmation récursive

On considère la suite de Fibonacci définie par : f[0]=0;f[1]=1;f[n]=f[n-1]+f[n-2]

4. La programmer de façon récursive. Que se passe-t-il si l'on demande f[100] ?
Taper et valider successivement `Flatten[Trace[f[10],f[3]]]` puis
`Flatten[Trace[f[10],f[3]]]//Length` : cette commande compte le nombre de fois où f[3] est recalculé quand on calcule f[10]

On peut résoudre le problème précédent en ne calculant qu'une seule fois chaque valeur de f. Pour cela, il suffit de la mémoriser par une affctation =, on écrira donc :
`f[n_]:=(f[n]=f[n-1]+f[n-2])`

5. Tester de nouveau f[100] et `Trace[f[10],f[3]]`

L'algorithme d'Euclide

Soient a et b deux entiers vérifiant : $a \geq b \geq 0$

Si $b \neq 0$, on effectue la division euclidienne de a par b : $a = bq + r$, q est noté `Quotient[a,b]` et r est noté `Mod[a,b]` en Mathématica

6. Programmer le calcul du pgcd de a et b de façon récursive en prenant $b = 0$ comme condition initiale (appelée plutôt condition d'arrêt en informatique) et en remarquant que si $b \neq 0$ alors :

$$pgcd(a, b) = pgcd(b, r)$$

7. Soit (u, v) un couple de coefficients de Bezout pour a et b, c'est à dire que u et v sont entiers et vérifient $au + bv = pgcd(a, b)$

Que peut on prendre pour (u, v) si $b = 0$?

Soit (u_1, v_1) un couple de coefficients de Bezout pour (b, r), trouver un couple de coefficients de Bezout (u, v) pour (a, b) en fonction de u_1, v_1 et q

8. Programmer le calcul d'un couple de coefficients de Bezout pour a et b de façon récursive

Les tours de Hanoï

On dispose d'un socle constitué de trois tiges. Sur la première tige, on place n disques de taille toute différente rangés du plus grand en bas au plus petit en haut.
Le but du problème est de déplacer tous les disques sur la troisième tige en respectant toujours la règle suivante : tout disque doit être placé soit sur un disque plus grand, soit sur le socle

9. Résoudre le problème à la main pour $n = 1, 2, 3$

10. (Sans Mathematica) Donner un procédé récurrent permettant de résoudre le problème, montrer que $2^n - 1$ déplacements sont nécessaires

11. Programmer la question 10, c'est à dire afficher les $2^n - 1$ messages, on donne l'initialisation : p, d, t signifie respectivement première, deuxième et troisième tige :
`hanoi[1,p_,d_,t_]:=Print["Déplacer le disque 1 de tige ",p," vers tige ",t]`