Programmer avec le logiciel Maple (2)

III Boucles itératives

III.2 Boucles while

Il n'est pas toujours possible de réaliser un calcul en un nombre prédéfini d'étapes. Il est parfois nécessaire de faire des opérations tant qu'une certaine condition est vérifiée. Par exemple, si l'on court un marathon, on ne s'arrête pas au bout de 35000 foulées sous prétexte qu'une foulée moyenne ferait 1,2m. On ne s'arrête pas tant que l'on n'a pas franchi la ligne d'arrivée. En mathématiques, il est souvent nécessaire de faire des calculs tant qu'une certaine propriété n'est pas vérifiée. Par exemple, si l'on souhaite calculer π à une précision donnée, il faut faire les calculs tant que l'on n'a pas atteint la précision voulue. En programmation, ce sont les boucles **while** qui vont permettre de réaliser de telles actions. Sous Maple, elles ont la syntaxe suivante :

```
> while condition do
    Instruction1 ;
    Instruction2 ;
    ...
od ;
```

où :

1. **condition** désigne un booléen.
2. **Instruction₁**, **Instruction₂**, ... sont des instructions Maple dépendant éventuellement d'une ou plusieurs variables.
3. **od** indique la fin de la boucle **while**.

Une telle commande répète les instructions **Instruction₁**, **Instruction₂**, ... tant que la condition **condition** est vérifiée.

Remarque III.1. Contrairement aux boucles **for**, on utilise une boucle **while** lorsqu'il est nécessaire d'enchaîner une suite de calculs identiques pendant un nombre d'étapes que l'on ne peut connaître préalablement. L'arrêt d'une telle boucle se produit lorsque la condition **condition** n'est plus vérifiée.

Remarque III.2. Il faut faire très attention au fait que si la condition est mal choisie, la boucle ne s'arrêtera pas. Le programmeur doit donc s'assurer lui-même que la boucle s'arrête dans tous les cas.

Exemple III.1. Comment déterminer le premier rang n tel que $\sum_{k=1}^n k$ soit supérieur ou égale à 7 ?

Nous mettons en oeuvre la technique vu lors du TP précédent pour élaborer un algorithme :

1. Il n'y a pas de paramètre au problème.
2. Nous devons calculer les termes consécutifs de la suite : $S_n = \sum_{k=1}^n k$ jusqu'à ce que $S_n \geq 7$.
3. Nous devons stocker dans une variable S le dernier terme de la suite calculer et dans k le rang auquel on est arrivé.
4. Avant d'écrire l'algorithme, remarquons que puisque nous n'utilisons pas une boucle **for**, il est nécessaire d'augmenter k d'une unité à chaque étape à l'aide d'une affectation.

Nous avons donc l'algorithme suivant :

$\left. \begin{array}{l} S := 0 \\ k := 0 \end{array} \right\} \text{ (initialisation)}$	
$\text{tant que } S < 7 \text{ faire}$	
$k := k + 1$	(on augmente k d'une unité)
$S := S + k$	(on calcule le rang suivant de la suite)
fin tant que	(à la fin de chaque itération de la boucle, la variable S contient S_k)
$\text{Afficher la valeur } k$	

5. Cela donne le programme Maple suivant :

```

> S := 0 ;
  k := 0 ;
  while S < 7 do
    k := k + 1 ;
    S := S + k ;
  od ;
  k ;

```

S := 0
 k := 0
 k := 1
 S := 1
 k := 2
 S := 3
 k := 3
 S := 6
 k := 4
 S := 10
 4

Il est possible de combiner une boucle for et une boucle while. Ainsi, la syntaxe générale d'une boucle itérative est la suivante :

```

> for var from ini to fin by pas while condition do
  Instruction1 ;
  Instruction2 ;
  ...
od ;

```

Une telle boucle itère la variable **var** depuis **ini** jusqu'à **fin** suivant le pas **pas** tant que la condition **condition** est vérifiée. A chaque itération, les instructions **Instruction₁**, **Instruction₂**, ... sont exécutées.

Exercice 1. Écrire un programme déterminant le premier rang n tel que $\sum_{k=1}^n k \geq M$.

Tester ce programme pour $M = 10$, $M = 20$ et $M = 100$.

Exercice 2. Écrire un programme déterminant le dernier rang n tel que $n! \leq M$.

Tester ce programme pour $M = 24$, $M = 119$ et $M = 721$.

Exercice 3. Nombres de Fermat.

Les nombres de Fermat sont les nombres de la forme $2^{2^n} + 1$ où n est un entier naturel. Fermat a conjecturé que ces nombres étaient premiers. Malheureusement, cette conjecture est fautive. Écrire un programme déterminant le premier rang n tel que $2^{2^n} + 1$ ne soit pas premier.

Indications : On pourra consulter l'aide de la fonction **isprime**.

Exercice 4. Suite de Syracuse (2)

La suite de Syracuse est définie de la manière suivante :

$$u_0 \in \mathbb{N}^* \text{ et } \forall n \in \mathbb{N} \begin{cases} u_{n+1} = \frac{u_n}{2} & \text{si } n \text{ est pair} \\ u_{n+1} = 3u_n + 1 & \text{si } n \text{ est impair} \end{cases}$$

Une conjecture affirme que, quelque soit le premier terme u_0 de la suite de Syracuse, il existe toujours un rang n_0 tel que $u_{n_0} = 1$.

Écrire un programme qui détermine le plus petit rang n_0 vérifiant cette propriété. Tester votre programme pour $u_0 = 10$, $u_0 = 32$ et $u_0 = 31$.

IV Les procédures

Nous venons d'écrire plusieurs programmes répondant à divers problèmes. Il serait intéressant de créer des instructions Maple qui, à partir des paramètres d'un problème, renvoient la réponse au problème.

Par exemple, nous voudrions créer une instruction `resol_2nd_degre` qui renverrait les solutions de l'équation $ax^2 + bx + c = 0$ à partir de la donnée de a , b et c . Les calculs s'effectueraient alors grâce à la commande `resol_2nd_degre(a,b,c)`.

IV.1 Définitions

Ces instructions sont appelées des procédures :

Définition IV.1. Une procédure est une suite d'instructions dépendant de variables d'entrée, appelées paramètres de la procédure, s'enchaînant les unes après les autres.

La création d'une procédure se fait comme suit :

```
[ > nom_proc := proc(arg1, arg2, ..., argm)
      local var1, var2, ..., varn ;
      global varn+1, varn+2, ..., varn+p ;
      instruction1 ;
      instruction2 ;
      :
      end ;
```

où :

1. $\text{arg}_1, \text{arg}_2, \dots, \text{arg}_m$ désigne les paramètres de la procédure.
2. $\text{var}_1, \text{var}_2, \dots, \text{var}_n$ désigne les variables locales de la procédure.
3. $\text{var}_{n+1}, \text{var}_{n+2}, \dots, \text{var}_{n+p}$ désigne les variables globales de la procédure.
4. `instruction1, instruction2, ...` sont des instructions Maple.
5. `end` indique la fin de la procédure.

L'exécution d'une telle commande crée une nouvelle instruction Maple portant le nom `nom_proc`. Elle se comporte comme n'importe quelle autre instruction Maple. Pour l'exécuter, il suffit d'exécuter la commande :

```
[ > nom_proc(arg1, arg2, ..., argm) ;
```

en remplaçant $\text{arg}_1, \text{arg}_2, \dots, \text{arg}_m$ par les valeurs désirées. Ces valeurs sont les valeurs d'entrée.

Maple exécute alors les instructions `instruction1, instruction2 ...` l'une après l'autre en fonction des valeurs des paramètres d'entrées; c'est à dire qu'il exécutera les instructions de la procédure en remplaçant $\text{arg}_1, \text{arg}_2, \dots, \text{arg}_m$ par les valeurs données en entrée. **Une fois exécutée, l'instruction `nom_proc` va renvoyer le résultat du dernier calcul effectué comme le ferait une fonction.** Cette valeur est la valeur de sortie.

Remarque IV.1. Lors de la création d'une procédure, on fera bien attention aux différents " ; ". En particulier, s'il n'y a ni variables locales ni variables globales, on fera suivre `proc(arg1, arg2, ..., argm)` d'un point-virgule. Dans le cas contraire, il ne faut pas en mettre.

Exemple IV.1. On veut construire la fonction définie de la façon suivante :

$$f : \begin{cases} \mathbb{R} \rightarrow \mathbb{R} \\ x \mapsto \sin(x) & \text{si } x < \frac{\pi}{4} \\ x \mapsto \cos(x) & \text{si } \frac{\pi}{4} \leq x \leq \frac{3\pi}{4} \\ x \mapsto -\sin(x) & \text{si } x > \frac{3\pi}{4} \end{cases}$$

La procédure suivante définit la fonction f :

```
[ > f := proc(x) ;
      if evalf(x < Pi/4) then
          sin(x) ;
      elif evalf(x <= 3*Pi/4) then
          cos(x) ;
      else
          -sin(x)
      fi ;
      end ;
      f := proc(x) if evalf(x < Pi/4) then sin(x) elif evalf(x <= 3 * Pi/4) then cos(x)
                  else - sin(x) end if endproc
```

Si nous voulons l'évaluer en $\frac{\pi}{12}$, il suffit d'exécuter l'instruction suivante :

```
[ > f(Pi/12) ;
                                     sin(π/12)
```

IV.2 Variables locales - Variables globales

Lors de la déclaration d'une procédure, on définit les variables locales et globales. Nous n'avons pas précisé quelle était la différence entre ces deux notions. Comme nous l'avons déjà vu précédemment, les variables servent, entre autre, à stocker des données, des résultats intermédiaires. . . Lorsque l'on crée une procédure, on va distinguer deux types de variables :

1. Les variables locales : elles ne sont modifiées qu'à l'intérieur de la procédure. Elle n'ont un "sens" qu'à l'intérieur de la procédure. En quelques sortes, elles ont un rôle similaire à celui des variables muettes que l'on rencontre dans les formules mathématiques (sommés, intégrales...). Son comportement est indépendant de celui d'une variable de la feuille de calcul qui porterait le même nom. En particulier, une variable de la feuille de calcul ne sera aucunement modifiée si l'on affecte une valeur à une variable locale qui lui est homonyme.
2. Les variables globales : au contraire, on ne distingue pas une variable globale d'une variable homonyme de la feuille de calcul. En particulier, lors de l'exécution de la procédure, la variable de la feuille de calcul de même nom est modifiée si l'on affecte une valeur à cette variable.

Quelle est l'intérêt d'une telle distinction ?

D'un côté, une variable locale peut porter le nom que l'on souhaite sans se soucier des conflits qu'engendreraient l'existence d'un homonyme. Ainsi, si quelqu'un utilise votre procédure, il n'a pas à connaître le nom des variables locales de cette procédure. De plus, elle permet de stocker le résultat d'un calcul intermédiaire que l'on ne souhaite pas conserver :

Exemple IV.2. Nous créons une procédure qui calcule $\sum_{k=1}^n k$. Pour cela, nous avons déjà vu qu'il suffit d'utiliser deux variables k qui indique à quelle étape du calcul nous nous trouvons et S qui contiendra la somme des termes jusqu'au rang k . Nous ne souhaitons pas utiliser ces variables en dehors de la procédure. Ce sont donc des variables globales de la procédure :

```
[ > Somme := proc(n)
      local S, k ;
      S := 0 ;
      for k from 1 to n do
        S := S + k ;
      od ;
      S ;
    end ;
    Somme := proc (n) local S, k; S := 0; for k to n do S := S + k end do; S end proc
```

Calculons $\sum_{k=1}^{100} k$:

```
[ > Somme(100) ;
                                     5050
```

Regardons le contenu de la variable S :

```
[ > S ;
                                     S
```

La variable S n'a pas été modifiée. En fait, la variable S de la feuille de calcul est différente de la variable S de la procédure. De plus, la variable S de la procédure a été effacée à la fin de l'exécution de la procédure.

D'un autre côté, les variables globales permettent, entre autres, d'altérer le comportement d'une procédure. Par exemple, nous avons déjà rencontré la variable `Digits` qui contient le nombre de chiffres significatifs utiliser lors des calculs avec les nombres flottants :

Exemple IV.3.

```
[ > evalf(Pi) ;
      Digits := 20 ;
      evalf(Pi) ;
                                     3.141592654
                                     Digits := 20
                                     3.1415926535897932385
```

On observe que le comportement de l'instruction `evalf` est altéré par la variable globale `Digits`. On peut également se servir d'une variable globale comme d'un compteur, par exemple, pour évaluer le nombre d'opérations faites lors d'une série de calculs.

Exemple IV.4.

```

> Cf := 0 ;
  f := proc(x)
    global Cf ;
    Cf := Cf + 1 ;
    sin(x) ;
  end ;
                                Cf := 0
                                f := proc(x) global Cf ; Cf := Cf + 1 ; sin(x) end proc
> map(f, [0, 1, 2, 3, 4, 5]) ;
  Cf ;
                                [0, sin(1), sin(2), sin(3), sin(4), sin(5)]
                                6

```

La fonction `f` calcule le sinus d'un angle tout en comptant le nombre de fois où l'on l'utilise. Ici, la commande `map(f, [0, 1, 2, 3, 4, 5])` ; fait appel à la fonction `f` à six reprises.

Exercice 5. Fonction valeur absolue

1. Créer une fonction valeur absolue de votre cru.
2. A l'aide de cette fonction, calculer $|- \pi|$, $|\ln(2 - \sqrt{3})|$.
3. Représenter graphiquement la fonction valeur absolue.

Exercice 6. En utilisant une variable locale bien choisie, créer la fonction suivante :

$$f : \begin{cases} \mathbb{R} \rightarrow \mathbb{R} \\ x \mapsto \tan^2\left(\frac{\pi}{4} \sin(x)\right) - \tan\left(\frac{\pi}{4} \sin(x)\right) & \text{si } x < 0 \\ x \mapsto \tan\left(\frac{\pi}{4} \sin(x)\right) & \text{si } 0 \leq x < \frac{\pi}{2} \\ x \mapsto \tan^2\left(\frac{\pi}{4} \sin(x)\right) - \tan\left(\frac{\pi}{4} \sin(x)\right) + 1 & \text{si } \frac{\pi}{2} \leq x < \pi \\ x \mapsto -\tan\left(\frac{\pi}{4} \sin(x)\right) + 1 & \text{si } x \geq \pi \end{cases}$$

On tracera son graphe sur $[-2\pi, 2\pi]$.

Exercice 7. Résolution des équations du second degré

1. Créer une procédure `Resol_2nd_degre` qui résout dans $\mathbb{R}$ l'équation $ax^2 + bx + c = 0$ à partir de la donnée de a , b et c .
Indication : Bien entendu, il est hors de question d'utiliser l'instruction `solve`.
2. A l'aide de cette procédure, résoudre les équations : $x^2 + 2x - 3 = 0$, $5x^2 + 4x + 22 = 0$ et $2x^2 + 4x^2 + 2 = 0$.

Exercice 8. Calcul d'une somme

1. Créer une procédure qui calcule $\sum_{k=1}^n k^2$ à partir de la donnée de n .
2. Calculer $\sum_{k=1}^{10} k^2$ et $\sum_{k=1}^{24} k^2$.

Exercice 9. Suites arithmétiques

1. Rappeler la définition d'une suite arithmétique.
2. Écrire, à l'aide d'une boucle, une procédure `arithm1` qui calcule le $n^{\text{ème}}$ terme d'une suite arithmétique de raison `a` et de premier terme `u0`. La procédure `arithm1` aura trois paramètres `n`, `a` et `u0`
3. Écrire une nouvelle procédure `arithm2` réalisant le même calcul sans utiliser de boucle.
4. Pour chacune des procédures précédentes, préciser en fonction de `n` le nombre d'opérations nécessaires pour effectuer le calcul.

Exercice 10. *Nombres premiers*

1. Écrire une procédure `estpremier` qui détermine si un nombre est premier ou non. Elle renverra `true` ou `false`.

Indication : On pourra utiliser le fait qu'un nombre est premier si, et seulement si, il n'est pas divisible par un nombre plus petit que $\sqrt{n}$. Toute autre optimisation pertinente sera la bienvenue.

2. Créer une fonction `PI` qui calcule le nombre de nombres premiers plus petits qu'un nombre x .
3. Représenter graphiquement la fonction `PI` et les fonctions d'expressions $\frac{x}{\ln(x)}$ et $\int_2^x \frac{1}{\ln(t)} dt$ sur l'intervalle $[0, 100]$ et sur un même graphique.