CONTRÔLE ET BOUCLES VI

Fonctions

1 Les booléens

On appelle **expression booléenne** une expression dont l'évaluation est **vraie** (*true*) ou **fausse** (*false*). Elles sont utilisées en particulier lors de tests et en programmation. Les expressions booléennes sont formées principalement de trois façons :

1.1 Opérateurs de comparaison

On les a déjà rencontrés. Ce sont :

```
=
<>
<
>
<=
>=
```

La comparaison de deux expressions par les opérateurs de comparaison est une expression booléenne. L'évaluation n'est pas effectuée en général. Il faut utiliser la fonction `evalb` pour forcer l'évaluation d'une expression booléenne.

Effectuer :

```
>x:=1;
>x=2;
>evalb(x=2);
>x>=0;
>evalb(x>=0);

>a:="chaîne":b:="chaine": # Bien noter la différence
>evalb(a=b);

>evalb(Pi>0);
>evalb((u+v)^2=u^2+v^2+2*u*v);
```

On note par ces deux derniers exemples les limites de la fonction `evalb`. Celle-ci opère une simple évalutation d'une expression booléenne, mais ne fait pas de calcul algébrique (quelques simplification évidentes cependant). On peut remplacer `evalb` par la puissante fonction `is`.

Reprendre les exemples précedents avec `is`.

1.2 Opérateurs logiques

Les expressions booléennes peuvent être combinées entre elles à l'aide des opérateurs logiques

```
not
and
or
```

Effectuer :

```
>r:=Pi/3; s:=Pi/6;
>not(sin(r)=cos(s));
>evalb((sqrt(2)/2=cos(r)) or (sqrt(2)/2=cos(s)));
>evalb(1/2=cos(r) or 1/2=cos(s));
```

```
>evalb(1/2=cos(r) and 1/2=cos(s));
```

Écrire les tables de vérité correspondant à ces trois opérateurs.

1.3 Fonctions à valeurs booléennes

Un certain nombre de fonctions donne pour résultat une expression booléenne. Ainsi `isprime`, déjà utilisée, renvoie des résultats booléens. La fonction `type` aussi. Elle indique si une expression est ou non d'un type donné. Voir l'aide pour plus de détail sur cette fonction.
Effectuer :

```
>type(Pi,realcons);
>type(Pi,string);
>type(Pi,symbol); >type(Pi,positive);
```

2 Structure de contrôle : `if`

On peut vouloir ne faire exécuter une commande que sous certaines conditions.

2.1 Cas d'une condition

La syntaxe à utiliser est :

```
if condition then instructions fi;
```

condition est une expression booléenne, et *instructions* une série de commandes séparées par des ; ou :. Les instructions ne sont exécutées que si l'évaluation de *condition* donne `true`.
Effectuer :

```
>r:=Pi/3; s:=Pi/6;
>if (cos(r)=sin(s)) then "Il y a égalité" fi;
>if (cos(r)<>sin(s)) then "Il n'y a pas égalité" fi;
```

2.2 Cas d'une alternative

La syntaxe à utiliser est :

```
if condition
     then instructions si condition est vraie
     else instructions si condition est fausse
fi;
```

Effectuer :

```
>t:=Pi/4;
>if (cos(t)=sin(t))
     then "Il y a égalité"
     else "Il n'y a pas égalité"
 fi;
```

Définir trois valeurs réelles `a,b,c` *et dire si les racines du polynôme* $aX^2 + 2bX + c$ *sont réelles ou complexes par une instruction* `if`.

2.3 Cas d'un choix multiple

La syntaxe à utiliser est :

```
if condition1 then instructions1
     elif condition2 then instructions2
⋮
     elif conditionN then instructionsN
     else instructionsN+1
fi;
```

Maple évalue *condition1*. Si le résultat est `true`, alors les *instructions1* sont exécutées. Sinon, il évalue la *condition2*. Si le résultat est `true`, alors les *instructions2* sont exécutées. Ainsi de suite. Si aucune condition n'est satisfaite, les instructions qui suivent `else` sont exécutées.

À vous :

1. *Reprendre l'exemple du polynôme de degré 2 en précisant le cas des racines doubles.*
2. *Réinitialisez, affectez la variable* `jour` *de la chaîne de caractères* `mercredi`. *Écrire une instruction utilisant la structure conditionnelle* `if` *donnant le menu du jour.* Exécuter de nouveau cette instruction après avoir changé l'affectation de `jour`.

3 Structure de boucle : `for` et `while`

3.1 Structure itérative simple : `for`

La syntaxe à utiliser est :

```
for i from début to fin by pas do instructions od;
```

La plupart des arguments sont facultatifs, sauf `do ... od`. Par défaut, *début* et *pas* valent 1.

Effectuer :

```
>for i from 1 to 10 do [i,evalf(sqrt(i),4)] od;
>k:=3;
>for k to 5 do k! od;
>k;
```

3.2 Structure itérative conditionnelle : `while`

La syntaxe à utiliser est :

```
while condition do instructions od;
```

Il s'agit à nouveau d'une boucle. Attention, il faut que les *instructions* modifie la condition, sans quoi la boucle est infinie !

Effectuer :

```
>restart;
>while n<100 do n:=n+rand(0..10)() od; # Corriger l'erreur.
```

3.3 Structure itérative complète

Les deux structures précédentes ne sont en fait que des cas particulier d'une structure itérative plus complète. Sa syntaxe est la suivante :

```
for ... from ... to ... by ... while ... do ... od;
```

À vous :

On appelle ***nombre de*** **FERMAT** *tout nombre qui s'écrit sous la forme* $2^{2^k}+1$. *Écrire une boucle calculant les nombres de* FERMAT, *qui s'arrête lorsque le nombre n'est pas premier, et donner la factorisation de ce nombre.*

4 Exercices

VI.1 Calculer 100! sans utiliser la touche [!] ni la fonction `factorial`.

VI.2 Déterminer le plus petit entier n tel que la somme des entiers de 1 à n dépasse 1000.

VI.3 Dresser la liste des 100 plus petits nombres premiers.

VI.4 Affecter à a et b deux valeurs. Échanger les deux affectations de a et b.

VI.5 Définir une liste u contenant 10 nombres. Écrire un algorithme permettant de ranger par ordre croissant ces 10 nombres.

VI.6 Un nombre entier n est dit parfait si la somme des diviseurs de n, distincts de n, est égale à n. Par exemple, 6 est parfait. Écrire un algorithme qui détermine si un entier n est parfait ou non. On pourra vérifier que 496 est parfait.

VI.7 Vérifier graphiquement que la fonction $x \mapsto \tan x - x$ admet un unique zéro sur $]\frac{3\pi}{2} - 1, \frac{3\pi}{2} - \frac{1}{10}[$. Déterminer une valeur approchée à 10^{-5} près de ce zéro en utilisant la dichotomie.

VI.8 **Algorithme de Syracuse**

L'algorithme de Syracuse est le suivant : Étant donné un entier n, on remplace n par $\frac{n}{2}$ lorsque n est pair et par $3n+1$ lorsque n est impair. On réitère alors le procédé jusqu'à obtenir 1.
Étant donné un nombre n, évaluer le nombre d'étapes nécessaires pour obtenir 1.

VI.9 (Voir exercice **6.24**)

Soit les suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ définies par $u_0 = a$, $v_0 = b$ et

$$\forall n \in \mathbb{N}, \quad v_{n+1} = \frac{u_n + v_n}{2} \qquad \frac{1}{u_{n+1}} = \frac{1}{2}\left(\frac{1}{u_n} + \frac{1}{v_n}\right)$$

On a vu qu'elles convergent vers la même limite appelée $\sqrt{ab}$.

1. *Question préliminaire* Rappeler le principe permettant de démontrer la convergence des deux suites, et le fait que ces deux suites convergent vers la même limite.
2. On prend $a = 1$ et $b = 2$ pour cette question. Calculer les valeurs prises par ces deux suites pour tous les entiers entre 1 et 20. Écrire une procédure permettant de calculer u_n et v_n ;
3. On donne $\varepsilon = 10^{-4}$. Déterminer une valeur approchée à ε près de la limite.

VI.10 (Voir exercice **6.21** et **6.22**)

Soit la suite dite **harmonique** définie pour $n \in \mathbb{N}^*$ par $u_n = \sum_{k=1}^{n} \frac{1}{k}$

1. Calculer u_{100} par un calcul direct (fonction `sum`) puis par une méthode itérative (boucle `for`) ;
2. On peut montrer que cette suite est croissante et diverge vers $+\infty$. On pose $A = 5$. Trouver N tel que $n \geqslant N \Longrightarrow u_n > A$.
3. Reprendre la question précédente en remplaçant A par 7.

VI.11 On va reprendre ici le résultat de l'exercice **6.17** , où l'on avait démontré que la suite définie par

$$S_n = \sum_{k=0}^{n} \frac{1}{k!}$$

est convergente de limite irrationnelle.

1. Calculer S_n pour des valeurs assez grandes de n et formuler une conjecture quant à la valeur de la limite, que l'on notera L.
2. On cherche maintenant à évaluer l'erreur $A_{100} = L - S_{100}$ de deux façons différentes. Faire un calcul direct (fonction `sum`) de A_{100} après avoir affecté à `Digits` une valeur adéquate.
3. Remarquer que
$$\sum_{k=1}^{100} \frac{1}{k!} = 2 + \frac{1}{2}\left(1 + \frac{1}{3}\left(1 + \frac{1}{4}\left(1 + \frac{1}{5}\left(\cdots\right)\right)\right)\right)$$
Calculer alors A_{100} par une méthode itérative (boucle `for`).
4. Comparer pour chacune des deux méthodes les résultats et les temps de calcul.