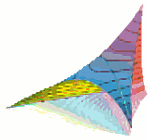

MI 101 Maple
Programmation
Feuille de TD numéro 4 v. 2 - jeudi 25 octobre

Table des matières

1 Infos diverses

Le programme des thèmes de TD est un peu modifié : après avoir repris la fin du TD 3, on verra cette semaine le débuts de la programmation (boucles et tests) sous Maple. Ce thème sera complété (procédures) au cours de la prochaine séance (le 08 novembre, car jeudi prochain est férié).

Et après 8, puis 6 et 5, le TD ne tient cette semaine que 4 pages, corrections comprises!!

2 Tests

2.1 Syntaxe booléenne

La syntaxe booléenne classique (oui/non, et/ou) est possible sous Maple. Sa première manifestation est les réponses de type 'true' ou 'false' que donne parfois Male, comme par exemple avec la fonction `isprime()`. Cette syntaxe est proche de celle du langage C ou C++, et correspond au sens commun du 'et/ou'.

Les tests booléens élémentaires sont les tests numériques:

- `a=b` renvoie 'true' si `a` et `b` sont égaux, 'false' sinon.
- `a<>b` renvoie 'true' si `a` et `b` sont différent, 'false' sinon.
- `a>b` renvoie 'true' si `a` est strictement supérieur à `b`, 'false' sinon. On peut aussi faire le test `a>=b`, pour 'supérieur ou égal'.
- `a<b` renvoie 'true' si `a` est strictement inférieur à `b`, 'false' sinon. On peut aussi faire le test `a<=b`, pour 'inférieur ou égal'.

Ces test élémentaires peuvent ensuite être combinés et manipulés; `X` et `Y` étant des tests semblables à ceux définis si-dessus,

- `(X)and(Y)` renvoie 'true' si `X` et `Y` sont vrais, 'false' sinon.
- `(X)or(Y)` renvoie 'true' si `X` ou `Y` sont vrais, 'false' sinon.
- `not(X)` renvoie 'true' si `X` est faux, 'false' sinon.

Les parenthèses ne sont pas toujours obligatoires, mais permettent une syntaxe plus claire dans le cas de tests compliqués.

Les fonctions Maple de tests, comme `isprime()` par exemple, qui renvoient 'true' ou 'false', peuvent aussi être manipulées et combinées.

On peut ainsi composer tous ces tests à loisir, comme

```
(isprime(a))or((irem(a,b)=0)and(not(isprime(b))))
```

qui teste si a est un nombre premier ou si b divise a et b n'est pas un nombre premier.

2.2 Instructions conditionnelles

Les tests permettent d'effectuer des instructions dans certaines conditions. la syntaxe type des instructions conditionnelles est :

`if` *condition* `then` *instruction1* `else` *instruction2* `fi;`

où *instruction1* est effectuée si *condition* est vraie et *instruction2* est effectuée si *condition* est fausse. La partie `else` *instruction2* est facultative. Si on ne la met pas, rien ne sera effectué si *condition* est fausse.

On peut imbriquer les tests, car *instruction1* ou *instruction2* sont des instructions Maple quelconques, qui peuvent contenir plusieurs commandes, des tests, des `if..then..else`, etc... il faut alors utiliser la fonction Male `print()` pour afficher une valeur, une variable,...

Il existe enfin un moyen plus simple d'effectuer plusieurs instructions conditionnelles que l'imbrication :

`if` *condition1* `then` *instruction1* `elif` *condition2* `then` *instruction2* `elif` *condition3* `then` *instruction3* `...else` *instruction* `fi;`

où *instruction1* est effectuée si *condition1* est vraie, *instruction2* est effectuée si *condition1* est fausse et *condition2* est vraie, *instruction3* est effectuée si *condition1* est fausse et *condition2* est fausse et *condition3* est vraie, ..., et *instruction* est effectuée si toutes les conditions sont fausses. Là encore, la partie `else` *instruction* est facultative.

3 Boucles

Plutôt que répéter un grand nombre de fois la même instruction, Maple permet de programmer une boucle, comme tout langage de programmation classique.

3.1 Boucles 'for'

La syntaxe-type est la suivante:

`for` *compteur* `from` *ini* `to` *fin* `by` *incrément* `do` *instruction* `od;`

où *compteur* est une variable, *ini* est la valeur initiale de *compteur*, *fin* sa valeur finale, *incrément* la valeur (positive ou négative) de l'incrémentation de *compteur* à chaque boucle et *instruction* est une instruction classique Maple. La partie `by` *incrément* est facultative, et l'incrément vaut 1 par défaut. *instruction* peut être composée de plusieurs commandes, à la suite, séparées par des `;` ou des `:`.

Lors de l'éxécution, Maple commence par indexer le compteur à sa valeur initiale, puis éxécute l'instruction et incrémente le compteur, jusqu'à parvenir à la valeur finale du compteur.

3.2 Boucles 'while'

La syntaxe-type est la suivante:

`while` *condition* `do` *instruction* `od;`

où *condition* est une condition booléenne quelconque et *instruction* est une instruction classique Maple. *instruction* peut être composée de plusieurs commandes, à la suite, séparées par des ; ou des :.

Lors de l'éxécution, Maple teste la condition, puis éxécute l'instruction et teste à nouveau la condition, jusqu'à ce que cette dernière ne soit pas vérifiée.

4 Exercices

4.1 Tests

Les premiers exercices sur tests sont élémentaires, et consistent à construire un test. Pour cela, il suffit de saisir la commande

`if` *test* `then print(``OK'');  else print(``probleme''); fi;`

où *test* est le test demandé, puis de valider cette instruction pour différentes valeurs des variables impliquées dans le test, afin de vérifier la validité du test.

Exo 1 Faites un test qui est vrai si et seulement si a est un multiple de 2, 3 ou 5.

Exo 2 Faites un test qui est vrai si et seulement si a et b, sont de même signe et c et d sont aussi de même signe ou si a et b, sont de signe différent et c et d sont aussi de signe différent.

Exo 3 Faites un test qui est vrai si et seulement si a, b et c sont des nombres complexes correspondant à des points équidistants (les modules de $a-b$, $b-c$ et $c-a$ doivent être égaux).

Exo 4 Construisez maintenant des instructions conditionnelles enchaînées telles que Maple réponde 1 si N vaut 2, 2 pour 3, 3 pour 5, 4 pour 7, 5 pour 9 et 6 pour 11, et 0 sinon.

Exo 5 DIFFICILE : construisez une instruction qui, si a est le produit de deux nombres premiers (test beaucoup utilisé en cryptographie, la 'science' du codage, de la sécurité et de la confidentialité des données), les affiche, et affiche 0 sinon. Etudiez pour cela le fonctionnement de la fonction `ifactors()`.

4.2 Boucles

Exo 6 Faites une boucle qui affiche les racines de x^n-1, pour n allant de 10 à 2, en ne prenant que les valeurs paires.

Exo 7 Stockez dans une liste les racines du polynôme $x^{20}-1$. A l'aide d'une boucle, effectuez la somme de ces racines. Pour cela, définissez `S:=0` avant la boucle, puis additionnez à `S` la iième racine au iième tour.

Exo 8 Pour un n de votre choix (assez grand), construisez une boucle qui remplace n par $n/2$ si n est pair ou par $n-1$ si n est impair jusqu'à ce que $n=0$, en affichant chacune de ces nouvelles valeurs. Pour tester la parité de n, utilisez `even()` ou `irem()`.

Exo 9 Faites la même chose que dans l'exo précédent, en stockant les valeurs successives de n dans une liste. Pour cela, définissez `L:=[]` liste vide avant la boucle, puis ajoutez à la séquence que contient `L` le iième résultat au iième tour.

5 Corrections

5.1 Tests

Exo 1

```
>if (irem(a,2)=0)or(irem(a,3)=0)or(irem(a,5)=0) then print(``OK''); else
print(``probleme''); fi;
```

ou plus malin...

```
>if (irem(a,2)*irem(a,3)*irem(a,5)=0) then print(``ca marche''); else print(``ca
marche pas''); fi;
```

Exo 2

```
>if ((a*b>0)and(c*d>0))or((a*b<0)and(c*d<0)) then print(``oh bah oui''); else
print(``ah bah non''); fi;
```

ou plus malin...

```
>if ((a*b)*(c*d)>0) then 1 else 0;
```

Exo 3

```
>if (abs(a-b)=abs(b-c))and(abs(b-c)=abs(c-a)) then 1 else 0 fi;
```

Exo 4

```
>if L=2 then 1
elif L=3 then 2
elif L=5 then 3
elif L=7 then 4
elif L=9 then 5
elif L=11 then 6
else 0 fi;
```

Exo 5 La fonction `ifactors(n)` renvoie une liste de deux éléments; le premier élément est 1 si $n > 0$, -1 si $n < 0$ et 0 si $n = 0$; le second élément est une liste des facteurs premiers; dans cette liste des facteurs premiers, chaque élément est un couple (une liste de deux éléments) dont le premier élément est le facteur et le second est son exposant. Il faut donc vérifier, par le résultat de ifactors, que n est > 0, qu'il est décomposé en deux facteurs et que les exposants de ceux-cis sont 1...

```
>L:=ifactors(a):
if op(1,L)<1 then print(0);
elif nops(op(2,L))<>2 then print(0);
elif (op(2,op(1,op(2,L)))=1)and(op(2,op(2,op(2,L)))=1) then
print(op(1,op(1,op(2,L))),op(1,op(2,op(2,L))));
else print(0); fi;
```

5.2 Boucles

Exo 6

```
>for n from 10 to 2 by -2 do solve(X**n-1); od;
```

Exo 7

```
>L:=[solve(X**20-1)]:  S:=0;
for i from 1 to nops(L) do S:=S+op(i,L): od:  S;
```

Exo 8

```
>n:=432:  while n>0 do if irem(n,2)=0 then n:=n/2:  else n:=n-1:  fi:  print(n);
od:
```

Exo 9

```
>n:=432:  L:=[]; while n>0 do if irem(n,2)=0 then n:=n/2:  else n:=n-1:  fi:
L:=[op(L),n]:  od:  L;
```