PROGRAMMATION 3

Boucles et tests

I - Sept manières de calculer 7!

1. Bien sûr, on peut commencer par faire

```
7!
```

mais le calcul de 7! n'est qu'un prétexte pour découvrir sur un exemple simple la syntaxe de programmation Xcas.

2. On pourrait alors taper

```
1*2*3*4*5*6*7
```

mais les choses pourraient se compliquer au moment de calculer 3232!

3. Nous allons donc utiliser une boucle *for* avec indice numérique : nous partons de 1, puis nous multiplions par 2, puis nous multiplions le produit précédent par 3, etc. Notons p le produit et k l'entier qui part de 1 et qui augment d'une unité jusqu'à atteindre 7. Cela donne

```
p:=1; for( k:=1; k<=7; k:=k+1)\{p:=p*k;\}
```

4. Nous aurions pu créer la liste des entiers de 1 à 7 puis utiliser une boucle *for* indexée cette fois par les éléments de la liste. C'est plus compliqué, mais cela nous permet de découvrir comment Xcas traite les listes. Il faudra bien distinguer

- les **ensembles** (*set*) qui sont des collections « *non classées* » d'expressions toutes différentes séparées par des virgules et encadrées d'accolades.

```
ens1:=set[2,4,1]; ens2:=set[2,5,8,5]
```

On peut effectuer les opérations usuelles sur les ensembles

```
ens1 intersect ens2; ens1 union ens2; ens1 minus ens2;
```

l'ensemble vide se note

```
set[]
```

- les **suites** (*sequence*) qui sont des collections « *classées* » d'expressions (c'est à dire avec un premier, un deuxième, etc.), différentes ou non, séparées par des virgules et encadrées ou non par des parenthèses.

```
5,7,5,1,2,3
```

On peut aussi utiliser les opérateurs seq et $ pour des suites définies par une formule explicite

```
seq(k^2,k=1..5)
```

```
(p^2) $ (p=1..5)
```

```
m$5
```

- ▷ les **listes** (*list*) qui sont des collections *classées* d'expressions séparées par des virgules et encadrées par des crochets. La différence, c'est qu'une suite, en tant que juxtaposition d'expressions, est en quelque sorte « en lecture seule », alors qu'une liste est une expression en elle-même et pourra donc « subir » des opérations algébriques.
  Notez au passage quelques fonctions utiles

```
s1:=(i)$(i=-2..2); s2:=a,b,c,d,e;

l1:=[s1]; l2:=[s2]; // une liste est une suite entre crochets

size(l2); //nombre d'opérandes

l2[3]; l2[0]; // notez bien que le premier opérande porte le
    numéro 0

l2[2..4]

select (x->x>0,l1); remove(x->x<0,l1);

 subsop(l1,2=32); subsop(l2,'3=NULL'); //pour substituer ou
     supprimer un opérande

map(l2,cos);

zip((x,y)->x*y,l1,l2);
```

Cela donne

```
rm_all_vars()}
l:=[k$( k=1..7)]; p:=1; for( k:=1; k<=7; k:=k+1){p:=p*l[k];}
```

5. Nous pouvons utiliser une boucle *while*

```
rm_all_vars()
p:=1; k:=1; while( k<7){k:=k+1; p:=p*k}
```

6. C'est bien beau, mais que ferons-nous quand il faudra calculer 32 ! ou 3232 ! et tous les autres ? Il faudrait créer un programme (on dira une **procédure**) qui donne $n!$ pour tout entier naturel n.

```
rm_all_vars() // je ne vous le dirai plus

fact(n):={\\ local p,k; // nous aurons besoin de variables locales i.e.
     internes à la procédure
  p:=1;
  for( k:=1; k<= n;k:=k+1){ p:=p*k ;}
} // termine la procédure
```

```
fact(32)
```

7. Le meilleur pour la fin : la **procédure récursive**, qui s'appelle elle-même

```
factr(n):={
  if (n==0){1} // un test $if...else$ pour régler le cas de 0!
  else{n*factr(n-1)}
}
```

```
factr(32)
```

En fait, ce mécanisme correspond à une suite numérique qui s'écrirait mathématiquement $u_n = n \times u_{n-1}$.

II - À vous de jouer...

Exercice 1 Partie entière

Déterminez une procédure `E(x)` qui, à un réel positif x, associe sa partie entière.

Exercice 2 Valeur absolue

Déterminez une procédure `ab(x)` qui, à un réel x, associe sa valeur absolue.

Exercice 3 Moyenne

Déterminez une procédure `moy(l)` permettant de calculer la moyenne des éléments d'une liste `l` de nombres réels. Il faut commencer par calculer la somme des opérandes de la liste puis diviser par le nombre d'opérandes de la liste.

Exercice 4 Équation du second degré

Déterminez une procédure `sol(a,b,c)` qui, à une équation $ax^2+bx+c=0$, associe son ensemble des solutions.

Exercice 5 test sur une liste

Déterminez une procédure `test(l)`, `l` étant une liste d'entiers, qui teste si ses éléments forment une suite croissante.