On souhaite parfois, lors d'une boucle, sauver à chaque tour une valeur. Il nous faudrait donc autant de variables que de tours de boucles, et une façon "automatique" d'appeler ces variables.
C'est dans ce but qu'existent les liste et les tableaux, objets que nous allons étudier au cours de ce TD. Les listes permettent de stocker un nombre non déterminé de variables dans un ordre linéaire, les tableaux permettent d'en sauver un nombre prédéterminé dans des cases multi-indexées.

1 Listes

1.1 présentation

Définition d'une liste : `[> L := [1,4,9,16];`
Liste vide : `[> L := [];`
Longueur d'une liste : `[> nops(L);`
i-ème élément de la liste : `[> L[i];` — i est un entier compris entre 1 et longueur(L)
Extraction d'une sous liste : `[> L[i..j];` — $1 \leq i \leq j \leq$ longueur(L)
Ajout d'un élément : `[> L :=[op(L),23];` — ajoute 23 à la fin de la liste
`[> L :=[op(L[1..i-1]),a,op(L[i..nops(L)])];` — ajoute a à la i-ème place
Définition "automatique" : `[> L :=[seq(k,k=1..5)];`
Opération sur tous les éléments : `[> map(x->x^2,L);`

Testez un peu les outils ci-dessus, afin de vous assurer que vous savez tous les utiliser. N'hésitez pas à aller regarder dans l'aide. Pour afficher les éléments d'une liste L les un après les autres, vous pouvez faire : `[> for i in L do print(i) od;`

Exercice 1 *Définissez la suite* $[1, 4, 9, 16]$ *de trois manières différentes (méthode directe,* `map`*,* `seq`*).*

Exercice 2 *Définissez* P*, liste des entiers pairs de 0 à 12. Supprimez le 3-ème élément de* P*.*

1.2 procédures utilisant des listes

Les exercices suivant ont pour objectif de vous entraîner à utiliser des listes dans des boucles, et de vous faire faire un peu de programmation. Il existe en général des fonctions Maple obtenant les mêmes résultats que les boucles suivantes.

Faites attention lorsque vous modifiez une liste dans une boucle ! Une commande du style : `for i to nops(L) do ... od;` sera comprise par Maple comme : pour i allant de 1 à la taille que fait L avant de commencer la boucle faire ... Il est en général bien mieux de créer une nouvelle liste qui pourra changer de taille dans la boucle mais dont ne dépend aucune condition d'arrêt.

Exercice 3 *Que fait l'algorithme suivant ? Programmez-le en Maple.*

```
Entreé : L
Variables locales : m, l, i
l := longueur(L)
m := L[1]
Pour i entre 1 et l faire
      si m<L[i] alors m := L[i] fin si
fin boucle

Renvoyer m
Fin
```

Exercice 4 *Écrivez une procédure qui dit si oui ou non un élément* a *est dans une liste* L*, et à quelle place (s'il y a plusieurs fois le même élément, vous pouvez renvoyer la place du premier, ou bien la liste des places de toutes les occurrences de* a*).*

Exercice 5 *Écrivez une procédure qui, à partir d'une liste* L *contenant des entiers naturels, renvoie une nouvelle liste contenant seulement les éléments non nuls de* L *(et dans le même ordre, bien sûr).*

Exercice 6 *Écrivez une procédure qui inverse l'ordre des éléments d'une liste.*

1.3 utilisation graphique

Lors du TD sur la méthode de Newton, il aurait été agréable de voir graphiquement la convergence de la suite. Le problème, c'est que les outils que vous utilisez pour les graphes (`plot`) ne permettent pas de tracer au fur et à mesure. Il faudrait donc pouvoir stocker les coordonnées (= liste de taille 2) de chaque point intéressant dans une liste.

Exercice 7 *Reprenez la procédure suivante et modifiez-la de façon à stocker les différentes valeurs de* `x` *(correspondant aux approximations successives de la racine de f) dans une liste* `X`*.*

```
[> Newton := proc(f,x0,N)
        local x,i ;
        x := x0 ;
        for i to N do x := evalf(x - f(x)/D(f)(x)) ;    od ;
        RETURN(x);
        end;
```

Si X contient les valeurs de la suite $(x_n)_{n\in\mathbb{N}}$, alors `F :=map(f,X)` contiendra les valeurs de la suite $(f(x_n))_{n\in\mathbb{N}}$. On désire tracer les segments reliant les points $(x_0, 0), (x_0, f(x_0)), \ldots, (x_i, 0), (x_i, f(x_i)), (x_{i+1}, 0), \ldots, (x_N, 0)$.
Pour tracer la ligne brisée passant par les points de coordonnées (x_i, y_i) pour i allant de 1 à n, on crée la liste suivante : `Coord := [ [`x_1`, `y_1`], [`x_2, y_2`], ...]` puis on utilise `plot` ainsi : `[> plot (Coord);`

Exercice 8 *Tracez sur un même graphe une fonction* f *donnée et les points correspondants à l'algorithme de Newton pour cette fonction.*

1.4 crible d'Eratosthène

Le crible d'Erathostène est un algorithme pour trouver les nombres premiers inférieurs à N. La méthode est la suivante :

- on se donne une liste des entiers de 2 à N ;
- on prend le plus petit élément de la liste, et on raye tous ses multiples ;
- on prend l'entier non rayé suivant et on recommence.

Exercice 9 *En vous inspirant de cette méthode, écrivez une procédure* `Crible(N)` *qui renvoie la liste des nombres premiers inférieurs à* N *(aide : au lieu de "rayer une case", on peut remplacer l'élément concerné dans la liste par 0, et pour finir se servir de l'Exercice 5).*

2 Tableaux

On a vu jusqu'ici des listes d'entiers. On peut en faire de lettres (3.3), de fonctions, de chats, de chiens, le tout dans une même liste, voir même des listes de listes (1.3). On pourrait imaginer aller plus loin encore, en créant des listes de listes de listes de... Il existe un outil spécial pour cela, lorsque à une profondeur donnée, toutes les listes ont la même taille (par exemple en dimension deux, si toutes les lignes ont le même nombre de colonnes).
On peut définir un tableau avec l'objet `array` de Maple, ce qui permet de l'utiliser plus facilement. En dimension deux, il s'agit d'une matrice, qu'on manipulera plus tard dans l'année.

```
[> M :=array(1..3,1..4);
[> M[2,3];
```

L'avantage, c'est qu'il existe des outils adaptés :

```
print(M);
```

Un autre avantage de array est que, sa taille étant fixée dès le début, Maple accepte l'utilisation de "grands" tableaux (essayez `Crible(500)` pour voir comme l'objet `liste` est limité).

Exercice 10 *Définissez la table de multiplication des entiers entre 1 et 10.*

3 Applications

Les algorithmes de tris sont des algorithmes que vous utilisez en permanence sans faire attention. Saurez-vous en programmez un ? Les plus rapides classent une liste de taille n en un temps proportionnel à $n.\ln(n)$. A quelle vitesse va le vôtre ?

3.1 algorithme intuitif

Comment classez-vous un jeu de carte ?

Exercice 11 *Écrivez sur une feuille de papier l'algorithme que vous emploieriez, avant de le programmer en Maple. Estimez ensuite la complexité de cet algorithme (= le nombre d'opérations nécessaires pour une liste de taille n).*

3.2 tri fusion

L'idée de départ de cet algorithme est qu'il est très rapide de faire une liste triée à partir de deux sous-listes déjà triées : on regarde les deux premiers éléments de chaque liste, on prend le plus petit, et ainsi de suite.

Exercice 12 *(facultatif) Écrivez une procédure `Fusion(L1,L2)` qui classe ainsi deux listes déjà triées.*

L'algorithme de tri est alors l'algorithme récursif suivant :

```
Tri(Tas) :
Variables locales : Tas1, Tas2
Si longeur(Tas)>1
        alors   Tas1 := première moitié de Tas
                Tas2 := seconde moitié de Tas
                Tas :=Fusion(Tri(Tas1),Tri(Tas2))
Fin si

Renvoyer Tas
Fin
```

3.3 listes de symboles

On peut transformer un mot ou une phrase (string) en une liste de lettres et symboles (char).

```
[> L :=[seq(l,l= "Bonjour!")];
[> M :=[seq(l,l= "f".."q")];
```

Exercice 13 *Vérifier si un mot donné est un palindrome (on pourra se servir de l'Exercice 6).*

Exercice 14 *(facultatif) Écrire les procédures `code(mot)` et `decode(mot)`, où le code consiste à remplacer a par b, b par c, ..., z par a.*