Benjamin COLLAS
benjamin.collas@telaxo.com

Mi-101 **Mathematica** 2004-2005

Leçon 2 : Les listes

Objectifs :
- lire et écrire la décompostion d'expressions mathématiques
- construire et manipuler des listes
- utiliser les fonctions booléennes sur les listes

Structure d'expressions mathématiques

■ Décomposition

Ces quelques exemples d'expressions vous permettront d'étudier le rôle de la structure en liste dans Mathematica.

```
expr1 = a + b + c;          expr5 = 18;
expr2 = a * x * b;          expr6 = 36 / 2;
expr3 = a + b * X + c * X^2; expr7 = 18.;
expr4 = {1, 2, 3, 4};       expr8 = 2 + I;
```

■ Exercice :

En utilisant les expressions précédentes ainsi que l'aide, découvrez quelle est la signification de *FullForm* et de *Head* .
La commande *Head* vous permet-elle d'expliquer en quoi *expr5, expr6* et *expr7* sont fondamentalement différentes ?
Cela est-il cohérent avec le partie *"précision finie et infinie"* de la *leçon 0* ?

■ Exercices :

Ecrivez la décomposition en arbre de la formules suivante $Cos(t + 2\frac{t}{Sin(t)})*Sin(3t) + \sqrt{2}$
Ecrivez l'expression tel que la fonction *TreeForm* renvoie :

```
Out[10]//TreeForm=
        Times[v, |                                        ]
                 Power[ |                              , -1]
                        Plus[1, |                    ]
                                Times[-1, |          ]
                                          Cos[teta]
```


■ Création par somme et multiplication

On considère ici les deux expressions *e1* et *e2* que l'on va additionner et multiplier entre elles.

```
In[1]:= e1 = a + b * X;
        e2 = c + d * X;
```

■ Exercices :

Utilisant l'addition et la commande *TreeForm*, trouver une règle de tranformation sur les arbres des expressions qui représente la somme.
De même pour la multiplication.

Créations et manipulations de listes

■ Création

```
lis1 = {a, b, {c, d}, d, e};
lis2 = List[a, b, {c, d}, d, e];
lis3 = Table[n, {n, 1, 10}];
lis4 = Range[10];
lis5 = Table[j, {j, 10}];
lis6 = Table[j, {j, 1, 10, .5}];
```

■ Exercice:

Etudiez ces différentes expressions et identifiez celles qui donnent le même résultat.

■ Exercice:

Construisez la liste des nombres entiers compris entre 0 et 5 de deux manières différentes.
Construisez la liste des nombres impairs compris entre 0 et 20.

■ Manipulations

Evaluez les lignes suivantes afin d'en déduire des règles de calculs sur les listes.

```
lis1 + lis2
lis1 * lis2
5 * lis1
a * lis2
```

Que se passe-t-il pour *lis1+lis2* et *lis1*lis2* si les listes n'ont pas le même nombre d'éléments ?

Voici maintenant quelques fonctions qui s'appliquent aux listes.
En les testant sur les exemples précédents, devinez leur signification.

```
Length      Reverse
RotateRight Drop
First       Join
Rest
```

Afin d'accèder au i-ème élément d'une liste, on peut utiliser la commande *[[i]]* accolée à celle-ci ou à son nom.
Etudiez les différentes syntaxes suivantes:

```
lis1[[1]]
{a, b, {c, d}, e, f}[[3]]
lis1[[3, 2]]
lis1[[3]][[2]]
lis1[[{1, 2}]]
```

■ **Exercice :**

Etudiez sur des listes les commandes *Union* et *Intersection*. Quelle différence y a-t-il entre *Join* et *Union*?

Une liste peut comporter des éléments qui sont eux-même des listes (*l1* en est un exemple) : elles ont plusieurs niveaux.
Découvrez l'utilité de la commande *Flatten* sur ce type de liste:

```
bino = Table[Binomial[n, p], {n, 0, 4}, {p, 0, n}]

Flatten[bino]
```

Qu'en est-il de la variante suivante ?

```
prof = Table[i + j + k + l, {i, 0, 2}, {j, 0, i}, {k, 0, i}, {l, 0, k}]

Flatten[prof, 1]

Flatten[prof, 2]

Flatten[prof, 3]
```

■ **Exercice :**

Prédire le résultat renvoyé par cette commande:

```
Table[RotateRight[lis4, i], {i, 0, 10}] // TableForm
```

■ Représentations

Les objets de type *"liste"* peuvent être représentés de différentes manières. De manière brute, comme les listes définies ci-dessus, ou de manière plus élaborées (en utilisant notamment *TableForm*).

■ **Exercice:**

Expliquez le résultat suivant:

```
lis1 + 3
liste1 = TableForm[lis1]

liste1 + 3
liste2 = lis1 // TableForm
liste2 + 3
```

Une autre manière de visualiser les listes consiste à les représenter sur un graph via la commande *ListPlot*.
Etudiez son utilisation sur les exemples suivants:

```
ListPlot[{1, 2, 3, 4, 5, 6}]

ListPlot[{-1, 2, 0, 4, -5, 6}]

Table[{Random[Integer, 10], Random[Integer, 10]}, {i, 1, 10}]

ListPlot[{{8, 5}, {6, 2}, {4, 7}, {0, 2}, {3, 4}, {1, 9}}]

ListPlot[{-1, 2, 0, 4, -5, 6}, PlotJoined → True]
```

■ **Exercice:**

Représentez graphiquement le cosinus des entiers compris entre 0 et 10.
Représentez graphiquement le cosinus des entiers compris entre 0 et 1000.

Interaction avec les fonctions

Dans cette partie vous étudierez des fonctions *appliquées aux listes* ou *utilisées avec elles*.

■ Fonctions listables

Certaines fonctions de *Mathematica* ont une propriété spéciale : lorsqu'elles sont appliquées à des listes, elles entrent automatiquement dans la liste afin de s'appliquer à chacun de ses éléments. On dit qu'elles sont *listables*.

Voici quelques exemples de fonctions listables:

```
Cos[lis1]

x^lis1

Power[x, lis1]
```

Dans le cas d'une fonction non listable, on peut tout de même l'appliquer à chacun des éléments de la liste en utilisant la commande *Map*.

■ **Exercice:**

La fonction *Length* est elle listable ?
En utilisant *Map* donnez la longueur du 3ème élément de la liste *l1*.

■ Fonctions booléennes

Une fonction booléenne est une fonction qui à un objet associe *True* ou *False.*

■ Exemples de fonctions booléennes

Pour tester une égalité on utilise le signe ==, les test d'inégalités larges sont exprimés par <= et >=.

```
1 == 2

And[3 ≥ 2, 3 ≤ 1]
Or[True, False]

EvenQ[2]

PrimeQ[3 * 5]

Function[x, x > 1][3]
```

■ Utilisation

Voici la principale commande qui permet de combiner fonction booléennes et listes.

```
Select

Select[Range[10], EvenQ]
```

■ Exercice :

En utilisant le modèle de la fonction qui renvoie *True* si *x* est supérieur à *1*, créez une fonction qui renvoie *True* si *x* est compris entre *1* strictement et *2* au sens large.

■ Exercice :

Calculez la liste des nombre premiers compris entre *1* et *20.*

Transformer une liste en expression mathématique

■ La fonction *Apply*

La commande *Apply* appliquée à une liste permet de changer son en-tête. Ainsi une liste de nombre peut facilement êre transformée en la somme de ces même nombres.

```
Head[lis4]
modif = Apply[f, lis4]
Head[modif]
```

■ Exercice:

En utilisant *Apply* et *lis4,* formez la somme puis le produit des entiers compris entre 1 et 10

■ Exercice:

Créez une fonction *sommeApprox* qui à *k* associe la somme des chiffres compris entre *1* et *k.*
Evaluez sa différence avec l'expression $\frac{n(n+1)}{2}$ pour différentes valeurs de *n*.
Qu'en déduisez - vous ?

■ Exemples d'approximations

Les outils développés dans cet feuille de TD permettent de tester différentes formules pour approcher des nombres tels que *e* ou π.

■ Approximation de *e* :

Calculez la somme $1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{n!}$ pour *n=10.*
Créez une fonction *eApprox* qui à *k* associe la différence entre e et la somme précedente jusqu'à *k.*
Appliquez la aux entiers compris entre 1 et 10 (on pourra utiliser une représentation graphique avec *PlotList*).
Que pouvez vous en déduire ?

■ Approximations de π :

Voici différentes formules pour approcher π. Vous devrez construire l'approximation puis discuter de la "bonne qualité" de celle-ci (faut-il beaucoup de termes pour avoir une bonne précision ?).

Formule de Gregory

La forme générale des termes que l'on somme est de la forme :

$$\frac{4 * (-1)^k}{2k + 1}$$

En se basant sur ce qui a été fait précédemment, construire la somme pour *k* allant de 0 à 10.
Créez une fonction *piGregory* qui à *k* associe la différence entre π et la somme précédente jusqu'à *k*.
Appliquez la aux entiers compris entre 1 et 100.

Formule d'Euler

La forme générale des termes que l'on somme est de la forme:

$$\frac{1}{p^2}$$

En se basant sur ce qui a été fait précédemment, construire la somme pour *k* allant de 0 à 10.
Créez une fonction *piEuler* qui à *k* associe la différence entre π^2 et *6* fois la somme précédente jusqu'à *k*.
Appliquez la aux entiers compris entre 1 et 100.
Comparez à l'approximation précédente.

Formule de Borwein et Plouffe

La forme générale des termes que l'on somme est de la forme:

$$\frac{1}{16^p} * \left(\frac{4}{8p+1} - \frac{2}{8p+4} - \frac{1}{8p+5} - \frac{1}{8p+6} \right)$$

En se basant sur ce qui a été fait précédemment, construire la somme pour *k* allant de 0 à 10.
Créez une fonction *piBorwein* qui à *k* associe la différence entre π et la somme précédente jusqu'à *k*.
Appliquez la aux entiers compris entre 1 et 10.
Comparez à l'approximation précédente.

De ces trois approximations de π quelle est celle qui d'après vous fournit le meilleur résultat ?