Université Bordeaux 1 — **Algèbre et calcul formel – Master**
Mathématiques — Année 2012–2013

FEUILLE D'EXERCICES n° 1

Travail sur machine : initiation à sage

1. Introduction

Sage est un logiciel libre de calcul, accessible sur http ://www.sagemath.org. Il utilise le langage python. Il existe trois façons de l'utiliser.

- En utilisant l'interface graphique (le notebook).
- Par la ligne de commande interactive, en appelant éventuellement des programmes écrits sur un éditeur de texte. Pour cela, il suffit de taper la commande `sage` sur une fenêtre terminal.
- En écrivant des scripts python indépendants qui font appel à la bibliothèque `sage`.

Nous utiliserons essentiellement la ligne de commande interactive.

2. Quelques commandes utiles

- Pour ouvrir `sage` : `sage` sur une fenêtre terminal.
- Pour quitter `sage` : `ctrl d`
- Pour interrompre un calcul : `ctrl c`
- Pour lire le programme toto.sage : `load` "toto.sage".
- Pour attacher le programme toto.sage : `attach` "toto.sage". Grâce à cette commande, à chaque fois que l'on tape `entrée`, le programme est relu. C'est utile si l'on a entre temps modifié ce programme.
- Dernier résultat : `_`
- Complétion : `Tab`
- Commentaire : # pour une ligne, ''' blablabla ''' pour un bloc.
- Informations sur une fonction : `Nom_de_fonction?` (par exemple `factor?`). Pour quitter l'aide ainsi activée, taper `q`.

Durant cette première séance, on pourra utiliser la ligne de commande interactive sans chercher à sauvegarder son travail. Par contre, à l'avenir, il conviendra d'utiliser un fichier ".sage", sous l'éditeur `emacs` que l'on attachera à la session de `sage`. Cela permettra de garder trace du travail effectué. Pour cette utilisation de `emacs`, il est commode de rajouter la ligne suivante dans votre fichier `.emacs` : cela permet de passer automatiquement en mode python lors de l'édition d'un fichier muni de l'extension `.sage`.

```
(setq auto-mode-alist (cons '("\\.sage$" . python-mode) auto-mode-alist))
```

3. Opérations de base, tests et affectations

Expérimenter les commandes suivantes (on tapera la touche `entrée` entre chaque commande).

```
1+1
a=8
a
a==5
a==8
parent(a)
type(a)
a=4/3
parent(a)
2<=3
```

(sage utilise = pour les affectations, et ==, <=, >=, <, > pour les comparaisons).

`2**5` (exponentiation)

`2^5` (autre écriture pour l'exponentiation)

```
10%3
10/3
10//3
4 * (10 // 4) + 10 % 4 == 10
```

Taper `a.`, puis actionner la touche `Tab`. Alors, la liste des fonctions pouvant s'appliquer à `a` apparaît.

4. Listes, ensembles, dictionnaires

Expérimenter les commandes suivantes.

```
S=[1,2,3,12]
len(S)
S[0]
S[3]
S[1]=6
S.append(10)
S.extend([18,19])
S[2:4]
S[2:]
S[:-1]
S=range(1..10)
S=[0..10]
S=[1,5,..33]
29 in S
S[1]=50
S.sort()
S=[i^2 for i in [1..10]]
S=[i^2+1 for i in [2,4..,10] if is_prime(i^2+1)]
```

`S=set([1,1,2,2,5,5,4,4])` (c'est un ensemble. Il n'y a pas de répétition et les entrées sont triées).

Dans sage, on peut définir des dictionnaires, c'est-à-dire des collections non ordonnées d'objets. On accède aux différentes valeurs classées dans le dictionnaire par des clefs. Taper par exemple les commandes suivantes.

```
dico={}
dico["clef"]=421
dico["seconde clef"]=0
dico["clef"]
```

5. Fonctions, nombres réels

Expérimenter les commandes suivantes.

```
sqrt(7.8)
sqrt(7)
n(_)
sin(pi/3)
sin(pi/3).n(digits=20)
n(pi,digits=10)
parent(_)
```

6. Polynômes

Dans sage, tous les valeurs sont définies en temps qu'éléments d'un ensemble bien précis, que l'on retrouve grâce à la commande `parent`, ou bien `type`. Nous avons vu l'exemple des entiers, des listes, des nombres rationnels et des réels.

Pour d'autres structures, il faut définir au préalable l'ensemble qu'on va considérer. C'est le cas par exemple pour les polynômes.

Expérimenter les commandes suivantes.

```
QQ
PR.<x>=PolynomialRing(QQ); PR
p=x^2-2; p
p=PR([-2,0,1]); p (c'est une autre façon de définir le même polynôme).
p.degree
p.is_irreducible()
factor(p)
p.roots()
P.roots(ring=RealField())
p(2); p[2]
p.coeffs()
```

7. L'anneau $\mathbb{Z}/n\mathbb{Z}$

Expérimenter les commandes suivantes.

```
A=IntegerModRing(12); A
A.list()
```

```
A.list_of_elements_of_multiplicative_group()
euler_phi(6)
a=A(5)
a^(10^10)
a.multiplicative_order()
```

8. Corps finis

On peut définir $\mathbb{F}_5 = \mathbb{Z}/5\mathbb{Z}$ de la même façon que ci-dessus. Mais il existe aussi une façon de définir tout corps fini. Pour $\mathbb{F}_5$:

```
k=GF(5)
```

Pour voir la liste des éléments du corps, taper :

```
k.list()
```

Pour définir $\mathbb{F}_{25}$, on tape la commande suivante.

```
k2.<t>=GF(25)
```

Taper

```
k2; k2.list()
```

On voit alors la liste des éléments de k2, comme polynômes en t. Pour connaître le polynôme minimal de t, taper

```
k2.modulus()
```

On peut aussi définir soi-même un générateur du corps. Taper les commandes suivantes.

```
PR.<y>=PolynomialRing(GF(5))
p=y^2+2; p.is_irreducible()
k3.b=GF(25,modulus=p); k3
```

9. Vecteurs, matrices

On peut définir un vecteur et une matrice de la manière suivante.

```
w = vector([1,1,-4])
A = Matrix([[1,2,3],[3,2,1],[1,1,1]])
```

Effectuer les opérations suivantes.

```
A.det()
A*w
w*A
kernel(A)
```

Remarquons que pour sage, le noyau d'une matrice A est le noyau à gauche, c'est-à-dire le noyau de l'application qui à x associe xA.

```
Y = vector([0, -4, -1])
X=A.solve_right(Y)
X; A*X
```

Si on veut travailler sur un corps fini, mettons $\mathbb{F}_5$, on peut définir vecteurs et matrices comme suit.

```
V=VectorSpace(GF(5),3)
w=V([1,2,3])
```

ou bien, de façon équivalente
`v=Vector(GF(5),[1,2,3])`
`MS=MatrixSpace(GF(5),3)`
`M=MS([1,2,3,2,1,0,0,1,2])`
ou bien, de façon équivalente
`M=matrix(GF(5),3,3,[1,2,3,2,1,0,0,1,2])`
Expérimenter les commandes suivantes.
`MS(0);MS(1)`
`M.nrows;M.ncols`
`M[0]; M[0,1]`

10. Boucles, fonctions

La syntaxe est celle de Python. La structure des fonctions est déterminée par l'indentation.

Pour créer une fonction `f` dont les indéterminées sont `a`, `b`, `c` :

```
def f(a,b,c):
  Instruction 1
  Instruction 2
  ⋮
  return ''résultat''
```

Expérimenter la fonction suivante, d'abord en suivant la ligne de commande, puis en l'écrivant sur un fichier à part `pair.sage`, à attacher grâce à la commande `attach`.

```
def pair(n):
  v = []
  for i in range(3,n):
    if i % 2 == 0:
      v.append(i)
  return v
```

Pour les boucles, on utilise la syntaxe suivante (la structure est toujours déterminée par l'indentation).

if :

```
if condition:
    instruction 1
    instruction 2
    ⋮
elif condition 2:
    instruction 1
    instruction 2
    ⋮
elif condition 3:
```

```
    instruction 1
    instruction 2
    ⋮
else:
    instruction 1
    instruction 2
    ⋮
```

while :

```
while condition:
    instruction 1
    instruction 2
    ⋮
```

for :

```
for i in L:
    instruction 1
    instruction 2
    ⋮
```

Ici, `L` peut être une liste, ou bien un n-uplet, ou bien un ensemble, ou une autre structure pouvant être énumérée.

Expérimenter par exemple :

```
k.<a>=GF(4)
for i in VectorSpace(k,2):
    print i
```

Sur un fichier à part, écrire une fonction qui étant donné deux entiers naturels m et n tels que $m \leqslant n$ rend la liste de tous les entiers de $\{m, \ldots, n\}$ divisibles par 8, 9 ou 7, rangés dans l'ordre croissant.