

FEUILLE D'EXERCICES n° 1 Initiation à Sage

1. INTRODUCTION

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

- En utilisant l'interface graphique, ou *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 le *notebook*

2. QUELQUES COMMANDES UTILES

- Pour ouvrir `sage` : `sage` dans une fenêtre Terminal, puis `notebook()`.
 - Pour quitter `sage` : fermer le notebook, puis `<Ctrl-d>`.
 - 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`. (Nous verrons d'autres moyens pour obtenir de l'aide.)

3. OPÉRATIONS DE BASE, TESTS ET AFFECTATIONS

Cliquer sur `Help`, puis `Work trough the tutorial`, puis `Assignment`, `equality and arithmetic`. Pour évaluer chacune des lignes, on peut, soit cliquer sur `evaluate` soit utiliser `shift enter`.

Après avoir lu cette page, ouvrir une session en cliquant sur `New worksheet`. Donner un nom à cette session, en suivant les instructions à l'écran.

Taper

```
a=9
```

et valider, puis taper

```
a.
```

et actionner la touche `Tab`. Alors, la liste des fonctions pouvant s'appliquer à `a` apparaît. Ici, `a` est un entier. Cliquer par exemple sur `a.N`. Normalement, cette expression apparaît sur votre ligne de commande. Si vous voulez savoir à quoi correspond cette commande vous validez

```
a.N?
```

Plus généralement, pour tout objet préalablement défini (par exemple un entier, un polynôme, un anneau, etc.), la complétion `Tab` permet de lister les méthodes qui peuvent lui être appliquées. C'est utile pour chercher le nom d'une fonction.

4. LISTES, ENSEMBLES, DICTIONNAIRES

1) Sur la même session, expérimenter les commandes suivantes (ne pas hésiter à évaluer S après chaque commande pour vérifier son effet).

```
S=[1,2,3,12]
len(S)
_ (dernier résultat obtenu)
S[0]
S[3]
S[1]=6
S.append(10) (ajout d'un élément à la liste S)
S.extend([18,19]) (concaténation)
S=S+[3,4,5] (autre syntaxe pour la concaténation)
S[2:4]
S[2:]
S[:-1]
S=range(1,10) (remarquer que l'on s'arrête à 9)
S=[0..10] (avec cette syntaxe, on va jusqu'à 10)
S=range(1,6,2) (le troisième argument est le pas de l'énumération)
S=range(6)
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 range(2,11) if is_prime(i^2+1)]
S=[[i,j] for i in range(4) for j in range(3)]
S=[[i*j for i in range(3)] for j in range(1,5)]
flatten(S)
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.)
```

Remarque. La commande `set(...)` donne l'ensemble défini par '`...`'. De même, `list(S)` produit la liste des éléments de l'ensemble S , c'est-à-dire ici la liste $[1,2,4,5]$. Plus généralement, si A est un objet, défini comme un ensemble, ou un ensemble muni d'une structure, comme un anneau, un corps, etc. et si a est un objet susceptible de définir un élément de A , alors la commande `A(a)` définit cet élément ; par exemple, l'entier 1 définit l'élément $\bar{1}$ de $\mathbb{Z}/3\mathbb{Z}$. Nous verrons d'autres exemples de ce principe plus tard.

2) Taper
`sum(S)`

La fonction `sum` additionne donc les éléments d'une liste, ou d'un ensemble. Elle peut être utilisée autrement :

```
sum(k^2,k,0,4)
```

Il y a un problème : il faut déclarer la variable k . Dans Sage, il faut toujours déclarer les objets sur lesquels on travaille. Faire plutôt

```
k=var('k'); sum(k^2,k,0,4)
```

En utilisant `sum`, retrouver l'expression de la somme des carrés des entiers compris entre 1 et n (pour obtenir la forme factorisée de cette expression, utiliser la fonction `factor`).

3) 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"]
```

4) Construire la liste $L = [1, 2, 3, 4, 5, 6]$. Quels sont les éléments $L[3]$, $L[0]$, $L[-1]$, $L[-3]$? Comment obtenir la liste $[2, 3, 4]$ directement à partir de la liste L ?

5) À l'aide de `range`, construire la liste des entiers compris entre 0 et 20. Que donne la commande `L[3:15:2]`? Construire la liste $[15, 14, 13, 12]$ à partir de L par une commande similaire.

5. BOUCLES, FONCTIONS

1) Dans le `tutorial`, faire les exercices de `fonctions`, `indentation and counting` jusqu'à `range(2,10)`.

2) On résume ci-dessous quelques commandes utiles pour la programmation. La syntaxe est celle de Python. La structure des programmes (le découpage en « blocs ») est déterminée par l'*indentation*.

- Pour créer une fonction `f` dont les paramètres formels sont `a`, `b`, `c` :

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

Expérimenter la fonction suivante.

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

- Pour les tests, on utilise la syntaxe suivante :

```

if condition:
    Instruction 1
    Instruction 2
    ...
elif condition 2:
    Instruction 1
    ...
elif condition 3:
    Instruction 1
    ...
else:
    Instruction 1
    ...

```

- Boucles :

```

while condition:
    Instruction 1
    Instruction 2
    ...

for i in L:
    Instruction 1
    Instruction 2
    ...

```

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

- 3) Expérimenter par exemple :

```

k.<a>=GF(3)
for i in VectorSpace(k,2)
    print i

```

4) Pour chacun des exemples suivants, écrire une fonction dépendant de n permettant de calculer la liste $L(n)$. Pour les quatre premières questions, on donnera deux méthodes : l'une utilisera une boucle, l'autre une liste « à formule » (c'est-à-dire du style `[i2 for i in range(n)]` pour la liste des carrés des entiers strictement inférieurs à n), ou une simple commande `range`.

- a) $L(n)$ est la liste des n premiers termes de la série harmonique

$$\sum_{i=1}^n \frac{1}{i}$$

- b) $L(n)$ est la liste des entiers impairs compris entre 1 et n .
c) $L(n)$ est la liste des n premiers entiers impairs.
d) $L(n)$ est la liste des entiers compris entre 1 et n qui ne sont divisibles ni par 2, ni par 3, ni par 5.

5) Calculer la somme de tous les multiples de 3 ou 5 inférieurs ou égaux à 1000.

6) On appelle triplet pythagoricien tout triplet (x, y, z) d'entiers naturels non nuls tels que

$$x^2 + y^2 = z^2.$$

a) Faire la liste de tous les triplets pythagoriciens dont toutes les composantes sont inférieurs ou égales à 50.

b) Déterminer les triplets pythagoriciens (a, b, c) tel que $a + b + c = 1000$ (il n'y en a qu'un, à permutation près de a et b).

7) Écrire une fonction qui calcule le n ème terme F_n de la suite de Fibonacci définie par : $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$ (on donnera deux fonctions : l'une sera récursive et l'autre utilisera une boucle).

8) Soit n un entier naturel. On note $E(n)$ l'ensemble des entiers naturels compris entre 1 et n .

a) Écrire une fonction qui, étant donné un entier n , donne la liste des parties de $E(n)$ à 2 éléments. Pour cela, on peut écrire la liste des $[i, j]$ tels que i est dans $E(n)$ et $i < j$.

b) Écrire une fonction qui, étant donnés n , k un entier tel que $1 \leq k < n$ et la liste des parties de $E(n)$ à k éléments (sous la forme $[i_1, \dots, i_k]$ où $i_1 < \dots < i_k$), donne la liste des parties de $E(n)$ à $k + 1$ éléments.

c) Écrire une fonction qui, étant donnés n et k , donne la liste des parties de $E(n)$ à k éléments.