TP d'informatique n°11
(Python Maths)

Utilisation du module Sympy de Python
DL, arithmétique et polynômes

L'objectif du TP est de manipuler les listes Python pour application aux polynômes.

On importera le module sympy. On utilisera les variables x comme variables formelles à l'aide de la commande

```
>>>  x, X = symbols('x'), symbols('X')
```

I Calcul de DL

Etant donnée une fonction f, on obtient son DL à l'ordre n (donné) et au point a (donné également) à l'aide de la commande :

```
>>>  series(f(x), x, a, n+1)
```

En effet le terme accompagnant le polynome est un O et non un o…

Si on veut récupérer uniquement le polynome, on utilise la commande :

```
>>>  series(f(x), x, a, n+1) . subs(O(x**(n+1)), 0)
```

Le module sympy possède une fonction plot permettant de tracer des courbes. Par exemple la commande

```
>>> plot(sin(x), series(sin(x),x,0,4).subs(O(x**4),0), (x,-pi,pi))
```

permet de tracer sur le même graphe la courbe de sin et de son DL à l'ordre 3 en 0, pour des valeurs de x comprises entre - π et π .

Tracer sur un même graphe la fonction exp et certains de ces DL en 0 à des ordres différents.
Effectuer le même travail avec la fonction x -> ln(1+x), arctan et d'autres fonctions usuelles…

II Arithmétique dans Z

1) **E**crire une fonction pgcdex prenant comme argument deux entiers relatifs a0 et b0, et retournant le pcgd de a0 et b0 ainsi que les entiers u et v obtenus par l'algorithme d'Euclide étendu et vérifiant : u a0 + v b0 = d où d est le pgcd de a0 et b0.

2) Tester votre fonction et résoudre les exercices de la feuille d'exercices d'arithmétique

3) Que font les fonctions du module sympy suivantes :

* igcd * ilcm * factorint

III Ecriture d'un polynôme, opérations sur les polynômes

Pour des raisons pratiques, un polynôme non nul $P = \sum_{k=0}^{p} a_k X^k$ donc avec $a_p \neq 0$, sera mémorisé dans Python par la liste des coefficients $[a_0, a_1, \ldots, a_p]$. Le polynôme nul sera symbolisé par la liste [0].

1. **E**crire une fonction coeff prenant comme argument une liste LP associée à un polynôme P et un entier naturel k, et qui renvoie le coefficient de degré k du polynôme. Attention si k > deg(P), coeff(LP, k) doit répondre 0.

2. **E**crire une fonction normalise prenant comme argument une liste L de flottants et qui rend la liste [0] si L est la liste vide ou n'est constituée que de 0, et la liste L débarrassée de ces 0 finaux sinon. La liste retournée est alors une liste associée à un polynôme.

3. **E**crire une fonction degre prenant comme argument une liste L de flottants et qui rend – 1 si le degré du polynôme associé à la liste L est nul et le degré de ce polynôme sinon.

4. **E**crire une fonction SommePoly prenant comme argument deux listes associées aux polynômes P et Q et qui rend la liste (normalisée) associée au polynôme P + Q. **C**alculer la complexité de cet algorithme en fonction du degré des polynômes.

5. **Si** vous n'êtes pas trop en retard, écrivez une fonction ProduitPoly prenant comme argument deux listes associées aux polynômes P et Q et qui rend la liste (normalisée) associée au polynôme $P \times Q$.

Comme on se rend vite compte que les écritures des opérations sur les polynômes risquent d'être un peu longues à mettre en place, on va utiliser le module sympy

IV Arithmétique des polynômes

Un polynome est créé à l'aide de la commande Poly, les multiplication et divisions euclidiennes de 2 polynomes sont obtenus par les fonction mul et quo et rem pour les quotient et reste. Testez également les fonctions ou méthode degree, coeffs, coeff_monomial.

1. **O**n rappelle l'algorithme d'Euclide étendu :

Données : a et b deux éléments d'un "anneau euclidien A" (comme $\mathbb{Z}$ ou $\mathbb{K}[X]$)
Initialisation : $u, v, r, u1, v1, r1 \leftarrow 1_A, 0_A, a, 0_A, 1_A, b$
Tant que $r1 <> 0_A$ Faire
$q \leftarrow$ Quotient de la division euclidienne de r par r1
$u, v, r, u1, v1, r1 \leftarrow u1, v1, r1, u - q * u1, v - q * v1, r - q * r1$ (les – et * étant les opérations dans l'anneau A et les affectations sont, ici, simultanées)
Fin Faire
Retour : (r, u, v)

Ecrire une fonction PGCD_etendu prenant comme argument deux polynomes A et B, et qui renvoie le tuple (D, U, V) associés respectivement au PGCD de A et de B, et aux polynômes U et V de Bézout de degré minimal vérifiant A U + BV = PGCD(A,B).

V Amélioration :

Des développeurs ont déjà répondu à cette recherche des PGCD et des polynômes de Bezout.

1. **A**près avoir chargé le module sympy, exécuter dans l'interpréteur :
   >>> x = symbols('x')
   >>> f , g = 2 * x**2 – 2*x - 4 , 2*x – 4
   Tester sur f et g les fonctions du module sympy suivantes : quo, gcd, rem, div, gcdex.
   Que font ces fonctions ?
   Effectuer les tests avec les polynômes (2X – 3, 2 X – 3.00000001), ($12X^2$ + 4X, 16 X^2), 3 autres couples avec des polynômes de degré de plus en plus grand…

2. **L**a fonction clock du module time donne "la valeur de l'horloge interne" au moment de l'appel. **C**omparer les durées d'exécution des fonctions PGCD_etendu et gcdex sur plusieurs exemples.

VI Exemples de planches d'Oral de concours :

Planche 142 / 2015 : Centrale MP

On rappelle que la fonction indicatrice d'Euler, notée φ, associe à l'entier n le nombre d'entiers de [0, n-1] premiers avec n.
On donne la fonction Python :

```
def listprem(n) :
    # Le but de cette fonction est de retourner une liste contenant l'ensemble des nombres
    # premiers avec n strictement plus petits que n
    if n == 1 :
        return([])
    resultat = [0] + [1 for i in range(n-1)]
    for k in range(2, n//2 + 1):
        if n%k == 0:
                for j in range(1, (n-1)//k+1):
                        resultat[j*k] = 0
    return([i for i in range(n) if resultat[i] == 1])
```

Calculer φ(n) pour n ∈{1, 10, p} où p est premier.

Montrer que la fonction **listprem** donne bien la liste voulue, en trouvant et en démontrant un invariant de boucle.
Ecrire une fonction **phi**(n) qui renvoie la valeur de l'indicatrice d'Euler appliquée à n.
Calculer phi(1024×81) et phi(1024) × phi(81) ; que conjecturer ? Le démontrer.

```
x = x + 1
if  y % 2 == 0 :
    y = 1 + y
x = x + y
```

Planche 270 / 2015 : ENSAM PSI

Les diviseurs propres d'un entier sont les entiers naturels qui lui sont strictement inférieurs et le divisent. Par exemple, pour 100 : 1, 2, 4, 5, 10, 20, 25, 50.

Ecrire une fonction **LDP** qui, pour $n \in \mathbb{N}$, renvoie la liste des diviseurs propres de n.

Ecrire une fonction **SDP** qui, pour $n \in \mathbb{N}$, renvoie la somme des diviseurs propres de n. (pour 100 on trouvera 117).
Un nombre parfait est égal à la somme de ses diviseurs propres ; écrire une fonction **parfaits**, qui pour $K \in \mathbb{N}$, renvoie la liste des nombres parfaits inférieurs ou égaux à K, en affichant au fur et à mesure "p est parfait".
Deux nombres sont dits amicaux lorsque l'un est égal à la somme des diviseurs propres de l'autre et inversement ; écrire une fonction **amicaux** qui pour $K \in \mathbb{N}$, renvoie la liste des couples (p,q) de nombres amicaux tels que $p < q \leqslant K$, en les affichant au fur et à mesure. Tracer les couples (p,q).