Premiers pas et structures de contrôle

Calcul formel – TP 1& 2

1. Nombre de chiffres d'un entier

1. Comme $10^{k-1} \leqslant N < 10^k$ alors $k-1 \leqslant \log N < k$, où log désigne le logarithme décimal, autrement dit $k-1 = E(\log N)$. Conclusion : $k = E(\log N)+1$ ce qui s'écrit `k = floor(log(N,10))+1`
2. **Énigme.** Voici une résolution élémentaire du problème posé. Une première boucle fait varier n, pour ce n développe $(x+y+z)^n$ puis une deuxième et troisième boucles énumèrent les coefficients de ce développement. Pour chaque coefficient, on affiche : n, le coefficient lui-même et son nombre de chiffres.

```
var('x,y,z')
nmax = 30

for n in range(2,nmax):
    P = expand((x+y+z)^n)
    for k in range(n+1):
        for l in range(n+1-k):
            C = P.coefficient(x^k*y^(l)*z^(n-k-l))
            print(n,C,floor(log(C,10))+1)
```

On obtient le premiers coefficient à 10 chiffres pour $n = 22$.

Réponse attendue : 22.

N'hésitez pas à améliorer cette fonction en utilisant un boucle `while` *pour que la recherche s'arrête dès qu'on atteint les 10 chiffres. Vous pouvez aussi étudier les coefficients binomiaux* `binomial(i,j)` *ou plus exactement des produits de coefficients binomiaux pour des valeurs des paramètres convenablement choisis.*

2. Constante de Champernowne

1. Pour construire la constante C_k, on part de C_{k-1} et on lui ajoute $k \cdot 10^{-i_k}$ pour un bon choix convenablement de l'exposant i_k. Celui-ci est le rang suivant $(i_{k-1}+1)$ lorsque k a un seul chiffre, par contre si k a deux chiffres il faut décaler de 2 : $i_k = i_{k-1}+2$ ...

```
C = 0
i = 0
for k in range(1,10):
    i = i + 1
```

```
    C = C + k*10^(-i)

for k in range(10,100):
    i = i + 2
    C = C + k*10^(-i)

print(C.n(digits=50))
```

2. **Énigme.**
   - Pour un réel $x = 1234,56789$ alors l'instruction `floor(x)` renvoie sa partie entière 1234.
   - Si on ne veut que les 5 derniers chiffres d'un entier $N = 12345678$ alors on calcule le reste de la division par 10^5 : `N % 10^5` renvoie 45678.
   - Pour décaler la virgule de k places vers la droite, on multiplie par 10^k. Ainsi pour $x = 1234,56789$ on a $x \cdot 10^3 = 1234567,89$.

   On applique toutes ces étapes à rebours :

```
a = 100
b = 102
n = floor(C*10^b) % 10^(b-a+1)
```

   donne l'entier 555 compris entre les rangs 100 et 102.

   On obtient la réponse à l'énigme en parcourant les résultats obtenus pour différentes valeurs de a.

   Réponse attendue : 166.

3. Nombres pseudo-premiers d'Euler

1. Voici le n-test d'Euler d'un entier p :

```
def test_euler(n,p):
    puiss = mod(n^((p-1)/2), p)    # n^(p-1)/2 modulo p
    if puiss == 1 or puiss == -1:
        return True
    else:
        return False
```

   Remarque : la façon dont on a calculé $n^{\frac{p-1}{2}} \pmod{p}$ n'est pas optimisée car on calcule un grand nombre et ensuite on le réduit modulo p. Il est préférable de remplacer la ligne correspondante par :

```
puiss = pow(n,(p-1)/2,p) # n^(p-1)/2 modulo p
```

2. **Énigme.** Il faut faire une boucle qui teste les entiers impairs jusqu'à obtenir un entier p non premier mais qui vérifie le 2-test d'Euler, le 3-test d'Euler et le 5-test d'Euler, c'est-à-dire la condition :

```
not(is_prime(p)) and test_euler(2,p) and test_euler(3,p) and test_euler(5,p)
```

   Réponse attendue : 1729.

4. Le nombre d'or

1. La commande `solve()` permet de résoudre l'équation $x^2 - x - 1 = 0$. On ne retient que la solution positive $\varphi = \frac{1+\sqrt{5}}{2} = 1,618\ldots$
2. Aucun problème, grâce à la commande `G = polar_plot(phi^( 2*t/pi ),(t,-2*pi,5*pi))` suivie de `G.show()`.
3. La suite est définie par la relation de récurrence $u_{n+1} = 1 + \frac{1}{u_n}$. Si (u_n) converge vers une limite ℓ alors la limite vérifie la relation $\ell = 1 + \frac{1}{\ell}$, autrement dit $\ell^2 - \ell - 1 = 0$. Ici la suite est positive, la limite est donc nécessairement φ. On renvoie à un cours d'analyse pour la preuve de la convergence.
4. **Énigme.** On pose `n = 18` et `u = phi^n`. Alors `u.expand()` renvoie $u_{18} = 1292\sqrt{5} + 2889$. Il est raisonnable d'essayer $a_{18} = 2 * 1292 = 2584$. On calcule alors $u_{18} - a_{18} \cdot \varphi$. On trouve alors $b_{18} = 1597$. On a ainsi obtenu $u_{18} = a_{18}\varphi + b_{18}$ avec $a_{18} = 2584$, $b_{18} = 1597$.

   Réponse attendue : 4181.

   Une preuve générale se fait par récurrence. Si on sait que $\varphi^n = a_n\varphi + b_n$ avec $a_n, b_n \in \mathbb{Z}$ alors on a
   $$\varphi^{n+1} = a_n\varphi^2 + b_n\varphi = a_n(\varphi + 1) + b_n\varphi = (a_n + b_n)\varphi + a_n,$$
   où on a utilisé que $\varphi^2 = \varphi + 1$. Ainsi $a_{n+1} = a_n + b_n \in \mathbb{Z}$ et $b_{n+1} = a_n \in \mathbb{Z}$ conviennent.
   Cela peut donner une autre méthode pour déterminer les coefficients a_n, b_n ou la somme $a_n + b_n$.